(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTIGRID APPROACH TO SOLVING THE LONG TRANSPORTATION PROBLEM ON A REGULAR GRID IN COST SPACE

by

Annette P. Cornett

June, 1993

Thesis Advisor: Van Emden Henson
Co-Advisor: Craig W. Rasmussen

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Master's Thesis

4. TITLE AND SUBTITLE

MULTIGRID APPROACH TO SOLVING THE LONG TRANSPORTATION PROBLEM ON A REGULAR GRID IN COST SPACE

5. FUNDING NUMBERS

6. AUTHOR(S)

CORNETT, Annette Pauline

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Multigrid methods were developed to solve partial differential equations. Research has shown that these methods are applicable to a broader range of problems. This thesis investigates the application of multigrid techniques to minimal cost flow problems, specifically the long transportation problem.

This research shows that multigrid techniques can be successfully applied to large-scale long transportation problems posed on a three-dimensional, regular grid in cost space. A V-cycle algorithm is developed for the long transportation problem. Analogies to the multigrid components of restriction, interpolation and relaxation are detailed. Performance of the algorithm is discussed, and computational cost is analyzed.

Future research is likely to include the development of more sophisticated restriction and interpolation schemes to provide integer-valued flows, and the development of a method to map an irregularly spaced problem to a regular grid, and to map the regular grid solution back to the original problem domain.

14. SUBJECT TERMS

Restriction, Interpolation, Multigrid Methods, Mimimal Cost Flow Problems

15. NUMBER OF PAGES

102

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

MULTIGRID APPROACH TO SOLVING THE LONG TRANSPORTATION PROBLEM ON A REGULAR GRID IN COST SPACE

by

Annette P. Cornett
Lieutenant, United States Navy
B.S., University of Southern Colorado, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June, 1993

Author: Annette P. Cornett
Annette P. Cornett

Approved by:
Van Emden Henson, Thesis Advisor

Craig W. Rasmussen, Co-Advisor

Maurice D. Weir, Acting
Richard Franke, Chairman
Department of Mathematics

ABSTRACT

Multigrid methods were developed to solve partial differential equations. Research has shown that these methods are applicable to a broader range of problems. This thesis investigates the application of multigrid techniques to minimal cost flow problems, specifically the long transportation problem.

This research shows that multigrid techniques can be successfully applied to large-scale long transportation problems posed on a three-dimensional, regular grid in cost space. A V-cycle algorithm is developed for the long transportation problem. Analogies to the multigrid components of restriction, interpolation and relaxation are detailed. Performance of the algorithm is discussed, and computational cost is analyzed.

Future research is likely to include the development of more sophisticated restriction and interpolation schemes to provide integer-valued flows, and the development of a method to map an irregularly spaced problem to a regular grid, and to map the regular grid solution back to the original problem domain.

THESIS DISCLAIMER

The reader is cautioned that the computer program developed during this research has not been tested on all cases of interest. While every effort has been made to ensure that there are no logic or computational errors, the program cannot be considered verified as yet. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. THE TRANSPORTATION PROBLEM 1

B. MULTIGRID METHODS 2

C. PREVIOUS RESEARCH 2

D. THESIS OVERVIEW 4

II. INTRODUCTION TO LINEAR PROGRAMMING 5

A. GENERAL LINEAR PROGRAMMING 5

B. MINIMAL COST FLOW PROBLEMS 7

C. THE TRANSPORTATION PROBLEM 9

D. THE SIMPLEX METHOD 10

E. THE NETWORK SIMPLEX ALGORITHM 13

F. AGGREGATION/DISAGGREGATION 15

III. AN INTRODUCTION TO MULTILEVEL TECHNIQUES 17

A. THE MODEL PROBLEM 17

B. BASIC ITERATIVE METHODS 19

C. FOURIER MODES 20

D. COMPONENTS OF MULTIGRID 22

1. Interpolation 25

2. Restriction 26

3. Coarse Grid Correction 26

4. The V-Cycle 28

5. Nested Iteration 30

6. The Full Multigrid V-Cycle 30

E. MULTIGRID METHODS APPLIED TO THE LONG TRANSPORTATION PROBLEM 31

IV. MULTIGRID COMPONENTS IN THE LONG TRANSPORTATION PROBLEM 33

A. BACKGROUND 33

B. RESTRICTION 35

C. INTERPOLATION 37

D. LOCAL SOLVER 39

E. RELAXATION 45

V. THE MODIFIED V-CYCLE 47

A. DESCRIPTION OF THE ALGORITHM 47

B. DATA STRUCTURE 50

C. COMPUTATIONAL COMPLEXITY 51

D. PERFORMANCE 52

VI. CONCLUSIONS AND RECOMMENDATIONS 55

A. CONCLUSIONS 55

B. AREAS FOR FUTURE RESEARCH 56

APPENDIX 58

LIST OF REFERENCES 90

INITIAL DISTRIBUTION LIST 91

LIST OF FIGURES

Figure 1. Illustration of Feasible Region, the Region in Which All Constraint Inequalities are Simultaneously Satisfied 6

Figure 2. Graphical Representation of a Mimimal Cost Flow Network 8

Figure 3. Graphical Representation of the Transportation Problem 10

Figure 4. One-dimensional grid on the interval $0 \leq x \leq 1$. Grid spacing is $h=1/N$. 17

Figure 5. Graphs of the Fourier modes on a grid with $N=12$, for wavenumbers $k=1, 4, 9$. 22

Figure 6. Initial guess (a) $\mathbf{w}_2$, (b) $\mathbf{w}_{12}$ and (c) combination of $\mathbf{w}_2$ and $\mathbf{w}_{12}$. The figures on the left show the approximation before iteration. The figure on the right is after ten iterations. 23

Figure 7. A wave with wavenumber k=4 on $\Omega^h(N=12)$ projected onto $\Omega^{2h}(N=6)$. 24

Figure 8. (a) Restriction of a fine grid vector to a coarser grid. (b) Interpolation of a coarse grid vector to a fine grid. 27

Figure 9. (a) The V-Cycle. (b) The FMV V-Cycle. The respective algorithms visit the various grids in the sequence indicated. 31

Figure 10. A Three Supply Node Problem Posed on a Regular Grid 34

Figure 11. (a) The global transportation problem on a regular grid. (b) The local transportation problem. 38

Figure 12. The V-Cycle Algorithm 48

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help of several people. I would like to take this opportunity to thank them now.

I would like to thank Wendi Patrick for her professionalism and her patience.

My gratitude to Craig W. Rasmussen and Van Emden Henson is immeasurable. Their guidance, encouragement, and patience is responsible for this thesis coming to fruition. They have my utmost respect and admiration.

My deepest thanks go to my husband, Marty. I could not have completed this thesis without his encouragement, love and support.

I. INTRODUCTION

A. THE TRANSPORTATION PROBLEM

The generic *transportation problem* contains m *origins* and n *destinations.* Origins and destinations are also referred to as *nodes* throughout this thesis. Each origin i contains a supply of s_i units of a particular commodity. Each destination j has a demand for d_j units of the same commodity. It is assumed that total supply equals total demand and that $s_i, d_j \geq 0$. The problem is to find a feasible shipping pattern that minimizes total cost. The problem is represented mathematically as follows:

$$
\begin{array}{llcccccccccl}
\text{Minimize} & z = c_{11}x_{11} + \dots + c_{1n}x_{1n} + c_{21}x_{21} + \dots + c_{mn}x_{mn} \\
\text{Subject to:} & x_{11} + \dots + x_{1n} & & & = s_1 \\
& & x_{21} + \dots + x_{2n} & & = s_2 \\
& & \cdot & & = \cdot \\
& & \cdot & & = \cdot \\
& & \cdot & & = \cdot \\
& & & x_{m1} + \dots + x_{mn} & = s_m \\
& x_{11} & + x_{21} & + x_{m1} & = d_1 \\
& \cdot & \cdot & \cdot & = \cdot \\
& \cdot & \cdot & \cdot & = \cdot \\
& \cdot & \cdot & \cdot & = \cdot \\
& x_{1n} & + x_{2n} & + x_{mn} & = d_n
\end{array}
$$

$$x_{ij} \geq 0 \quad i=1,2,\dots,m, \quad j=1,2,\dots,n$$

In general, a transportation problem with m supply nodes and n demand nodes contains mn variables and $m+n$ constraint equations.

A transportation problem where $m<<n$ is referred to as the *long transportation problem*.

B. MULTIGRID METHODS

Multigrid methods evolved out of an effort to improve iterative solution methods for boundary value problems. The continuous boundary value problem is transformed into a discrete problem by choosing a set of grid points in the domain for the problem. At each grid point, an approximation to the differential operator is formed using the difference between values of the unknown function at neighboring grid points. This leads to a system of linear equations relating the values of the unknown function at the grid points. A one-dimensional problem is used in this thesis to illustrate multigrid methods for ease of demonstration, although multigrid techniques are more commonly applied to higher dimensional problems.

C. PREVIOUS RESEARCH

Aggregation/disaggregation is a method applied to large scale transportation problems. It involves consolidating several neighboring origins or destinations to form a smaller problem that is easy to solve, and provides an initial guess for the original problem. This method aggregates demand nodes until a small problem, which is easy to solve, is obtained. The solution is then disaggregated to provide

an initial solution or a bound for the original problem. Kaminsky (1989) developed a multilevel algorithm that uses an approach similar to the aggregation/disaggregation approach. Kaminsky extended this approach from two levels to multiple levels. He required that the supply and demand nodes occupy a physical location in space and that shipping costs be directly related to the physical distance between a supply node and a demand node. He then aggregated nodes together that were physically close.

Cavanaugh (1992) relaxed this geometric restriction so that shipping costs may have no correlation to the physical distance between the supply and demand nodes. He then mapped the demand nodes to what he called *cost space,* where the shipping cost from each supply node determines its position. For example, cost space for the *mxn* problem is the m-dimensional space in which each coordinate axis represents the shipping costs from each of the m supply nodes. Each of the n demand nodes are then placed in cost space at the point whose coordinates are the shipping costs from the supply nodes to it (Cavanaugh, 1992). This generally results in a transportation problem posed on an irregular grid in cost space. The idea is to aggregate the demand nodes together that have similar costs.

This thesis uses the idea of cost space and assumes that the problem is posed on a regular grid.

D. THESIS OVERVIEW

The algorithm developed in this thesis is not intended to replace or compete with any of the existing solution methods. The transportation problem is being used as a test bed, and an effective method would, after development, be applied to a class of more intractable problems.

Chapter II provides background information on the transportation problem and outlines traditional solution methods. Chapter III provides an introduction to multigrid techniques and describes the key elements in multigrid methods. Chapter IV describes the analogies to the key elements in multigrid methods developed in this thesis. Chapter V describes the entire algorithm and contains the results of the numerical experimentation. It also describes the storage requirements and provides a complexity argument of the algorithm. Chapter VI contains conclusions and recommendations for future research.

II. INTRODUCTION TO LINEAR PROGRAMMING

A. GENERAL LINEAR PROGRAMMING

Linear programming is concerned with the *optimization* of a linear function with n variables subject to m linear constraints. The general form of the linear programming problem is as follows:

$$\text{Minimize} \quad \Sigma_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{Subject to:} \quad \Sigma_{j=1}^{n} a_{ij} x_j \geq b_i \quad i=1,2,\ldots,m \tag{2}$$

$$x_j \geq 0 \qquad j=1,2,\ldots,n \tag{3}$$

The *objective function* is represented by the summation in (1) and is usually denoted by z. The term optimization refers to the minimization or maximization of the objective function. A minimization problem can be easily transformed into a maximization problem and vice versa by simply taking the negative of the objective function as follows: $\text{Max}\Sigma_j c_j x_j = -\text{Min}\Sigma_j(-c_j x_j)$. The *cost coefficients* are $c_1, c_2, \ldots, c_n$ and the decision variables are $x_1, x_2, \ldots, x_n$. The *constraint inequalities* are represented by (2). The coefficients a_{ij} for $i=1,\ldots,m$ and $j=1,\ldots,n$ are referred to as the *technological coefficients*. Inequality (3) is the nonnegativity constraint. The inequality constraints can be easily transformed into equalities by adding a *slack* or *surplus* variable $x_{n+1} \geq 0$. Hence $\Sigma a_{ij} x_j \geq b_i$ can be rewritten as $\Sigma a_{ij} x_j - x_{n+1} = b_i$,

likewise $\Sigma a_{ij}x_j \leq b_i$ can be rewritten as $\Sigma a_{ij}x_j + x_{n+1} = b_i$. A set of values for $x_1, x_2, \ldots, x_n$ satisfying all the constraints is referred to as a *feasible point, feasible vector,* or a *basic solution,* although it may not yield an optimal value for the objective function. The set of all such points is known as the *feasible region* (see Figure 1). If the solution contains fewer than *m+n-1* positive solution variables then the solution is said to be *degenerate.*

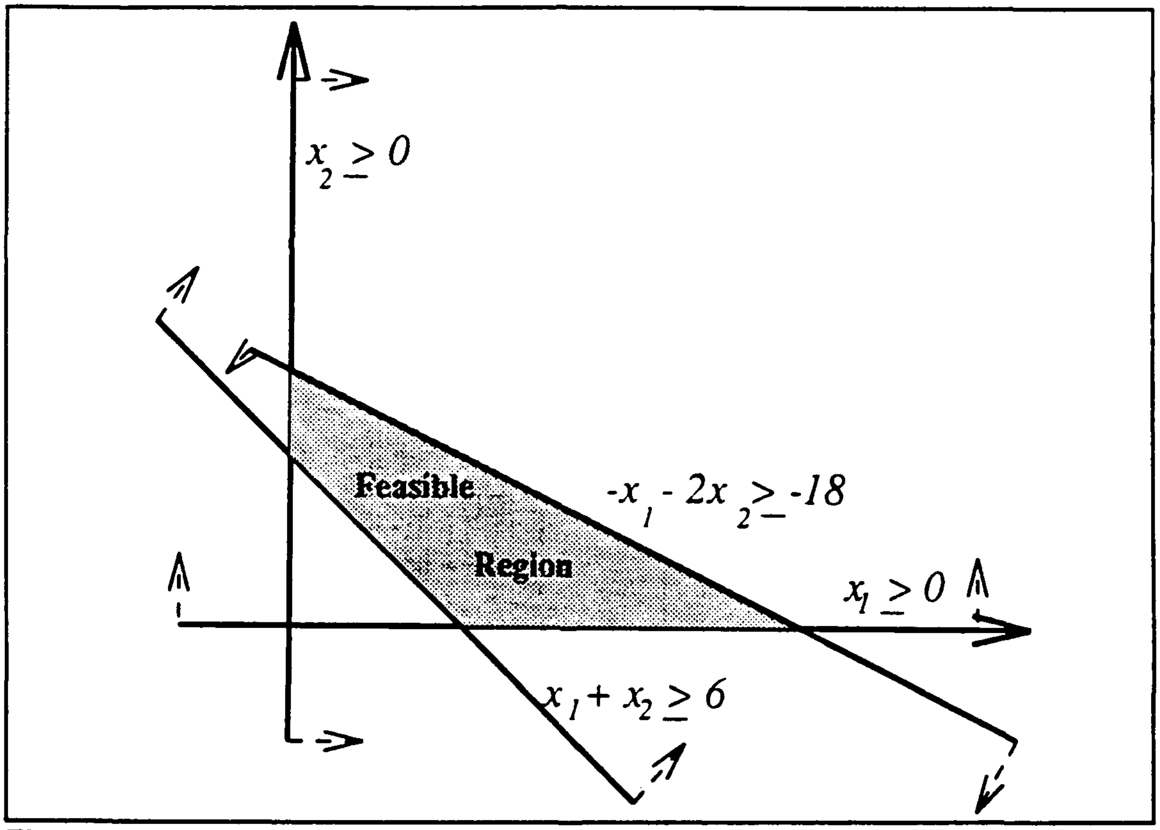

Figure 1. Illustration of Feasible Region, the Region in Which All Constraint Inequalities are Simultaneously Satisfied

The problem can be presented in matrix notation as follows:

$$\begin{aligned} &\text{Minimize} \quad cx \\ &\text{Subject to:} \quad Ax = b \\ &\qquad\qquad\qquad x \geq 0 \end{aligned}$$

$$c = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Note that the inequality constraints have been transformed into equality constraints; this is standard practice, since systems of equations are easier to solve than are systems of inequalities.

B. MINIMAL COST FLOW PROBLEMS

A *graph* consists of a set V of m *nodes* (or vertices) and a set E of *edges*, which are unordered pairs of elements from V. A *directed graph*, or *digraph*, D, consists of a set V of m nodes and a set A of *arcs* which are ordered pairs of elements from V. The nodes are represented by points and there is a directed line from i to j if and only if $i,j \in V$ and $(i,j) \in A$. A *network* is a directed graph in which every node i has a number b_i associated with it, where $i=1,\ldots,m$, and each arc has a capacity for flow that is greater than or equal to zero (see Figure 2). The number

b_i represents either the supply of some commodity present at the node or the demand of the commodity, that is, the amount required at the node.

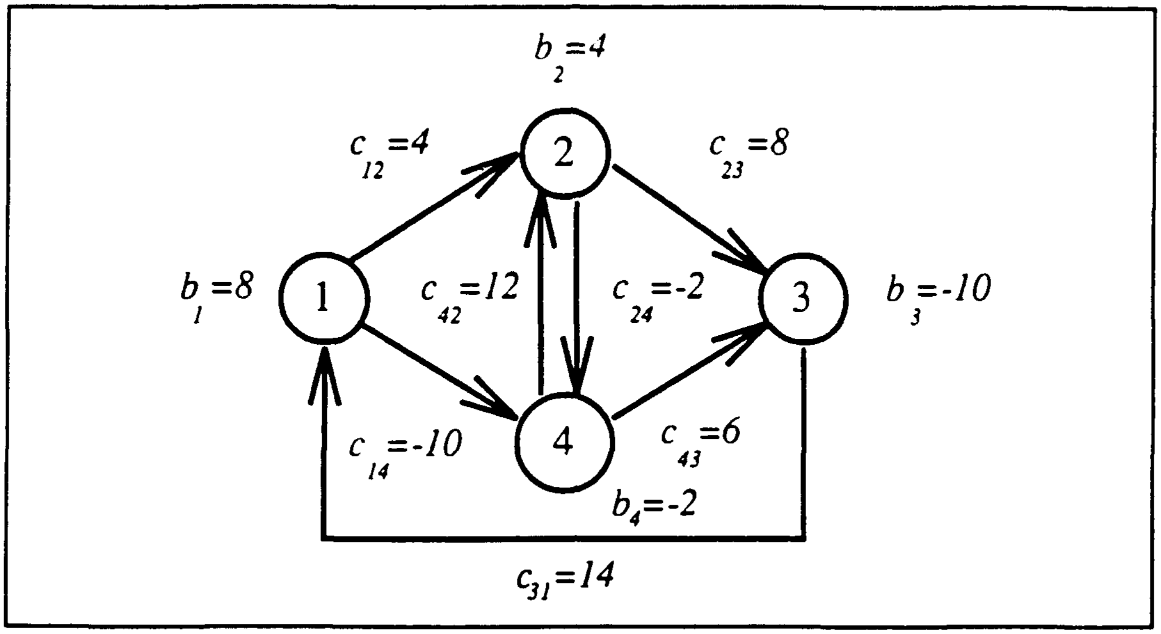

Figure 2. Graphical Representation of a Mimimal Cost Flow Network

The *minimal cost flow problem* is to ship the available supply through the network to satisfy the demand at the cheapest possible cost. Let m represent the total number of nodes in the network. Let x_{ij} represent the flow along the arc from node i to node j. Let c_{ij} represent the cost to ship one unit of flow along the arc (i,j). Both x_{ij} and c_{ij} are assumed to be nonnegative. If $b_i<0$, then b_i represents the demand at node i, while if $b_i>0$, then b_i represents the supply at node i. The underlying assumption in this problem is that total supply equals total demand, therefore $\sum_i b_i=0$. If this is not the case then a *dummy node* can be added to the network so that $b_{m+1}=-\sum_i b_i$, and then arcs with zero cost from each supply node to

the dummy node b_{m+1} would also be added. The general minimal cost flow problem can be written mathematically as follows:

$$\begin{aligned} &\text{Minimize} && \sum_i \sum_j c_{ij} x_{ij} && \\ &\text{Subject to:} && \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i && \forall i \qquad \textbf{(4)} \\ & && x_{ij} \geq 0, && i,j=1,2,\ldots,m \end{aligned}$$

Equation (4) is referred to as the *conservation of flow equation*. $\sum_j x_{ij}$ represents the total flow out of node i, and $\sum_k x_{ki}$ represents the total flow into node i. Because the network is actually a directed graph, solution techniques can take advantage of this special structure to solve these problems much faster than general linear programming problems.

C. THE TRANSPORTATION PROBLEM

The transportation problem is the simplest case of the minimal cost flow problem. It can be represented by a *complete bipartite digraph* (see Figure 3). This digraph is *bipartite* because the set of nodes can be partitioned into two disjoint sets, O={origin or supply nodes} and D={destination or demand nodes}, and all arcs are directed from O to D. The digraph is *complete* because there is an arc connecting every node in O to every node in D. If an actual problem does not contain an arc connecting some node $i \in O$ to some node $j \in D$, then an artificial arc is added to the graph and assigned high cost so that x_{ij} becomes an artificial variable.

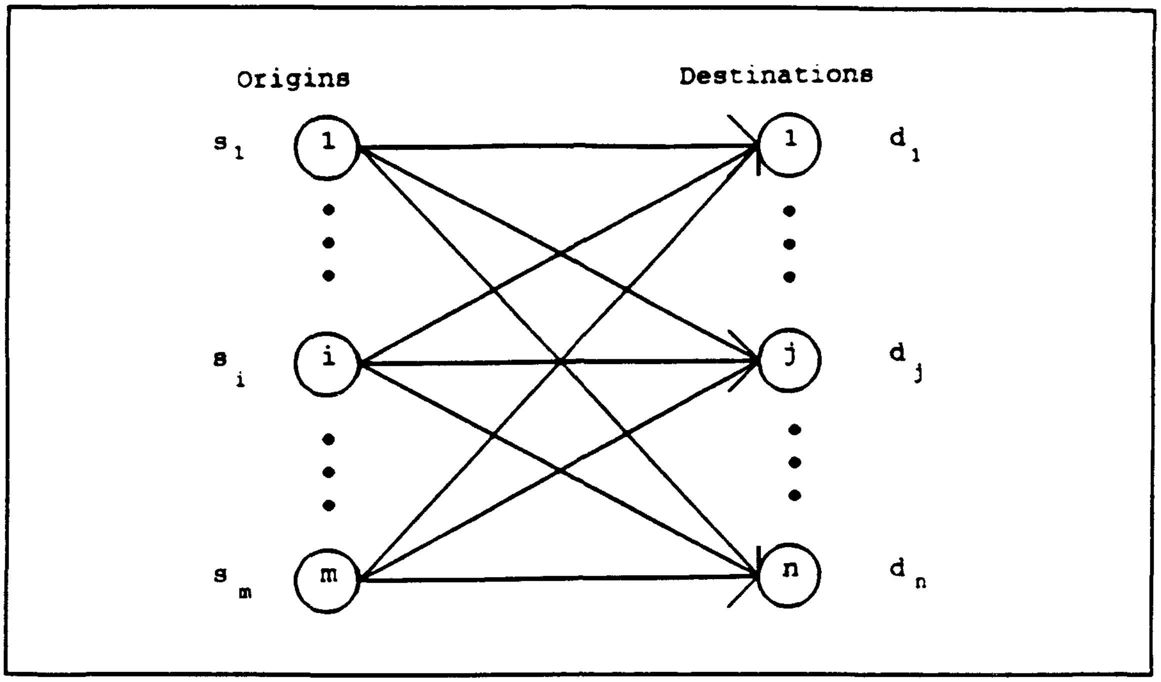

Figure 3. Graphical Representation of the Transportation Problem

The assumption that total supply equals total demand guarantees the existence of a feasible solution to the transportation problem, provided that no constraints are placed on the carrying capacity of the arcs. For example, it can be shown that $x_{ij}=(s_i d_j)/d$ is a feasible solution, for all $i=1,\ldots,m$, $j=1,\ldots n$, where $d=\Sigma_j d_j=\Sigma_i s_i$. Also, a bound exists for each decision variable, such that, $0 \leq x_{ij} \leq \text{minimum}\{s_i, d_j\}$. It is well-known that a bounded linear program with a feasible solution has an optimal solution (Bazaraa, 1990). Some traditional solution methods are presented next.

D. THE SIMPLEX METHOD

Whenever feasible solutions to a linear programming problem exist, the region formed by the constraint equations is called the *feasible region* (see Figure

1). It can be shown that this region is also *convex*, which means that each pair of points in the region can be joined by a line segment that lies within the region. The boundaries of this region are lines or planes and there are corners where these lines or planes intersect. Points on these corners are referred to as *extreme points*. Two extreme points are *adjacent* if the line segment joining them is an edge of the feasible region. Any point in the region can be written as a convex combination of the extreme points, which leads to the property that an optimal solution can be found at an extreme point.

Informally, the simplex method can be summarized as follows:

1. Begin at an initial extreme point.
2. Perform an optimality test (Is this extreme point at least as good as the adjacent extreme point(s)?). If yes, stop; optimality has been achieved.
3. If not, move to an adjacent extreme point whose objective function value is at least as good as the present extreme point. Repeat step 2.

The simplex method, as presented above, is an algorithm used to solve exactly a linear programming problem in a finite number of steps or reveal that the solution is *unbounded*, that is, that the objective function has a value of $-\infty$ and hence no optimal solution exists. This method breaks a difficult problem into a series of easy problems. Some notation must be introduced before the simplex method can be formally presented.

Consider the system $\mathbf{Ax=b}$, $\mathbf{x \geq 0}$. Let $\mathbf{A}$ be an mxn matrix, where $m \leq n$, and $\mathbf{b}$ a vector of length n, and suppose that the rank of the augmented matrix $[\mathbf{A,b}]$

satisfies rank ($[\mathbf{A},\mathbf{b}]$)=rank($\mathbf{A}$)=m. The matrix $\mathbf{A}$ can be partitioned into $\mathbf{A}=[\mathbf{B}\ \mathbf{N}]$, where $\mathbf{B}$ is an mxm invertible matrix and $\mathbf{N}$ is an $mx(n\text{-}m)$ matrix. A solution to the system is $\mathbf{x}^T=[\mathbf{x}_B\ \mathbf{x}_N]$, where $\mathbf{x}_B= \mathbf{B}^{-1}\mathbf{b}$ and $\mathbf{x}_N=\mathbf{0}$. This is referred to as a *basic feasible solution.* $\mathbf{B}$ is the *basic matrix* and $\mathbf{N}$ is the *nonbasic matrix*. The columns of $\mathbf{B}$ are also referred to as the *basis*. The vector $\mathbf{x}_B$ contains values of the *basic variables* and $\mathbf{x}_N$ contains values of the *nonbasic variables*. If $\mathbf{x}_B>\mathbf{0}$ then the solution is *nondegenerate,* while if at least one component of $\mathbf{x}_B$ equals zero then the solution is *degenerate*. The cost vector is partitioned so that $\mathbf{c}=[\mathbf{c}_B\ \mathbf{c}_N]$, where $\mathbf{c}_B$ contains the basic variable costs and $\mathbf{c}_N$ contains the nonbasic variable costs. The *dual variables* are contained in the row vector $\mathbf{u}$, where $\mathbf{u}$ is given by $\mathbf{u}=\mathbf{c}_B\mathbf{B}^{-1}$. The dual variables are used to compute the *reduced costs* denoted by $(z_j\text{-}c_j)=\mathbf{u}\mathbf{a}_j\text{-}c_j$, where $\mathbf{a}_j$ is the jth column of the matrix $\mathbf{A}$ and z_j is given by $z_j=\mathbf{c}_B\mathbf{B}^{-1}\mathbf{a}_j$ for each nonbasic variable. The reduced costs are actually the cost coefficients of the nonbasic variables. The simplex algorithm for a minimization problem is as follows:

1. Start with a basic feasible solution with basis $\mathbf{B}$. (There are many different procedures available for finding an initial basis.)

2. Solve the system $\mathbf{B}\mathbf{x}_B=\mathbf{b}$, which has the unique solution $\mathbf{x}_B=\mathbf{B}^{-1}\mathbf{b}$. Let $\mathbf{x}_N=\mathbf{0}$ and $z=\mathbf{c}_B\mathbf{x}_B$.

3. Solve the system $\mathbf{u}\mathbf{B}=\mathbf{c}_B$, which has the unique solution $\mathbf{u}=\mathbf{c}_B\mathbf{B}^{-1}$. Calculate $z_j\text{-}c_j=\mathbf{u}\mathbf{a}_j\text{-}c_j$, for all nonbasic variables. Let $z_k\text{-}c_k=$ maximum$(z_j\text{-}c_j)$ for all $j\in R$ where R is the current set of indices associated with nonbasic variables. If $z_k\text{-}c_k\leq 0$ then stop, the solution is optimal. Otherwise x_k is the *entering variable.*

4. Solve the system $\mathbf{B}\mathbf{y}_k=\mathbf{a}_k$, with unique solution $\mathbf{y}_k=\mathbf{B}^{-1}\mathbf{a}_k$. If $\mathbf{y}_k\leq\mathbf{0}$, that is, if all components of the vector $\mathbf{y}_k$ are negative, then the optimal solution is unbounded. Otherwise continue.

5. The variable x_k enters the basis, while x_{Br} leaves the basis. The index r of the variable x_{Br} which leaves the basis is determined by the minimum ratio test. If the columns of $\mathbf{B}$ are the basis then $\mathbf{a}_k$ enters the basis while x_k becomes one of the basic variables. The minimum ratio test is as follows:

$$\frac{x_{Br}}{y_{kr}} = minimum_{1\leq i\leq m}\left\{\frac{x_{Bi}}{y_{ki}} : y_{ki}>0\right\},\ r\in R$$

6. Update the basis $\mathbf{B}$, where $\mathbf{a}_k$ replaces $\mathbf{a}_{Br}$, update the set R, and repeat step 2.

A transportation problem with 25 supply nodes and 1000 demand nodes consists of 25000 variables and 1025 constraints. This type of problem cannot be solved efficiently or quickly using the simplex method, even with the aid of a large scale computer. The *network simplex method* is suited to the task because it exploits the unique structure of the graphical representation of the transportation problem.

E. THE NETWORK SIMPLEX ALGORITHM

Several terms must be defined before the network simplex method can be discussed. A digraph D contains a ***path*** from s to t if an ordered set of the arcs in A begin at s and end at t, where $s,t\in V$. That is, the arcs (w_0,w_1), (w_1,w_2), $(w_2,w_3),\ldots,(w_{n-1},w_n)$ where $w_0=s$ and $w_n=t$, and w_j are all in A, for $j=0,1,2,\ldots n$. A path

that begins and ends at the same node is a *cycle*. A digraph is *connected*, if between every pair of vertices s and t there is a path. A *tree* is a connected digraph with no cycles. A *spanning tree* in D is a tree containing every node in D. A *rooted tree* has one node identified as the root such that there exists a unique path from the root to every node in D.

If the solution to a minimal cost flow problem is examined graphically it corresponds to a spanning tree in the network (Bazaraa, 1990). The network simplex method exploits this fact and is a more efficient solution method than the simplex method. The network simplex algorithm is as follows:

1. Find an initial basic feasible solution represented by a rooted spanning tree.
2. Compute the basic flows, x_B, and the dual variables u_i.
3. For each nonbasic arc (i,j), compute the reduced cost u_i-c_{ij}-u_j.
4. If u_i-c_{ij}-$u_j \leq 0$ then stop, as the solution is optimal. Otherwise select a nonbasic arc (i,j) with a positive reduced cost to enter the basis (append to the current basic tree).
5. Divert flow to the entering arc until flow along another arc is reduced to zero. Remove the arc with zero flow from the tree, thereby creating a new basic spanning tree. Go to step 2.

This method can be applied directly to the actual network representation of a small problem and it can be programmed to solve larger and more complicated problems. Recall that the transportation problem is the simplest case of a minimal cost flow problem. Large scale transportation problems are normally solved using

the network simplex method, but occasionally a method known as aggregation/disaggregation is utilized.

F. AGGREGATION/DISAGGREGATION

Aggregation/disaggregation (referred to as aggregation) is a method used to solve large scale transportation problems. It involves partitioning the variables in a problem and replacing each partition with a single variable. The same technique is applied to the constraints. The resulting problem is referred to as an aggregate problem. The method involves consolidating several "neighboring" origins or destinations to form a much smaller problem that is easier to solve and that will provide insight into the solution to the larger problem. The term "neighboring" is intentionally loosely defined so that there are not explicit restrictions on the aggregation method (Balas, 1963).

Aggregation provides an approximate solution to the original problem while considering only a small portion of the problem data. To solve the original problem directly using network simplex with m origins and n destinations requires mn evaluations of the reduced costs, while if the aggregation method is used the number of computations decreases significantly, depending on how many aggregate variables are used. For instance, if a problem containing m=350 origins and n=500 destinations is solved directly it will contain mn=175,000 variables and 850 constraint equations. If the aggregation method is used and aggregation is five by five (five neighboring origins become one origin and five

neighboring destinations become one destination), then there are M=70 aggregated origins and N=100 aggregated destinations, resulting in MN=7000 variables and 170 constraint equations. The gain in computational savings increases with the size of the problem and the concentration of the origins and destinations. This method is not widely used, because the accuracy of a solution is often traded for computational savings, and an optimal solution is therefore not guaranteed. It must be emphasized that while the nontraditional solution approach to the long transportation problem in this thesis is related to the aggregation method, there are significant differences between them.

III. AN INTRODUCTION TO MULTILEVEL TECHNIQUES

A. THE MODEL PROBLEM

Consider the boundary value problem which describes the steady state distribution of temperature in a long uniform rod. This may be represented by the second order differential equation $-u''(x)+\sigma u(x)=f(x)$, where $0<x<1$, $\sigma\geq 0$ (5) and subject to the boundary conditions $u(0)=u(1)=0$ (Briggs, 1987). This equation will be referred to as the model problem.

Occasionally, the model problem can be solved analytically, but it is more common to solve it numerically. Multigrid methods are applied to numerical solution techniques. One such numerical method of solution is the finite difference method. The domain $0\leq x\leq 1$ is partitioned into N subintervals with spacing $h = 1/N$, and each point is labeled x_j, where $x_j = jh$. This is the finest grid and is denoted by Ω^h (see Figure 4).

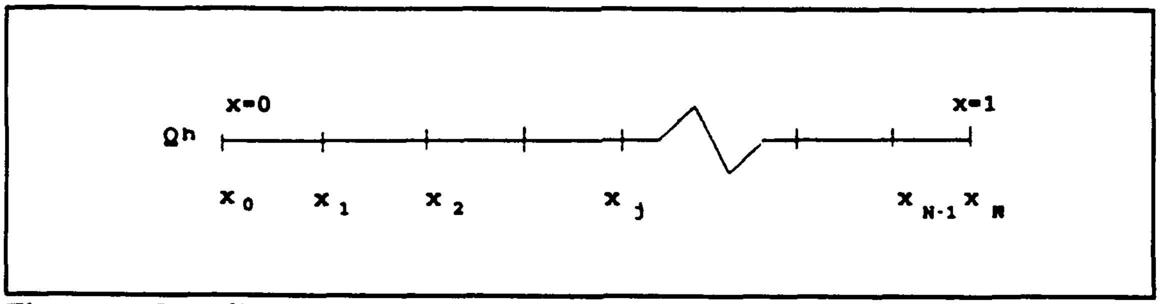

Figure 4. One-dimensional grid on the interval $0\leq x\leq 1$. Grid spacing is $h=1/N$.

At each grid point the differential equation is replaced with a finite difference equation. Let $u(x_j)$ represent the exact solution and $v(x_j)$ represent the approximation to the exact solution, where $1 \leq j \leq N-1$. For ease of notation let u_j represent $u(x_j)$ and v_j represent $v(x_j)$, and let $\mathbf{v}=(v_1, v_2, \ldots, v_{n-1})^T$. We seek to find $\mathbf{v}$ such that the components of $\mathbf{v}$ satisfy $N-1$ linear equations, each having the form $(-v_{j-1}+2v_j-v_{j+1})/h^2+\sigma v_j=f(x_j)$, where $1 \leq j \leq N-1$. The boundary conditions yield $v_0=v_N=0$. This discretization may be represented in matrix notation as

$$\frac{1}{h^2}\begin{bmatrix} 2+\sigma h^2 & -1 & & & \\ -1 & 2+\sigma h^2 & -1 & & \\ & & \cdot & & \\ & & \cdot & & \\ & & \cdot \ \ldots & -1 & \\ & & -1 & 2+\sigma h^2 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_{N-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_{N-1} \end{bmatrix}$$

or simply as $\mathbf{Av} = \mathbf{f}$ (Briggs, 1987). Note that the matrix $\mathbf{A}$ is tridiagonal, sparse and symmetric positive definite with dimensions $(N-1)x(N-1)$. This system could be solved directly using Gaussian elimination, but this method is computationally expensive. Indirect methods such as *iteration* can also be used to solve systems of this type. A special case of iteration is *relaxation*. Jacobi and Gauss-Seidel are two examples of relaxation methods. Relaxation begins with an initial guess at the solution, and then tries to improve the current approximation through a series of iterations. Ideally this sequence of approximations will converge to the exact solution. The rate of convergence of basic iterative methods for certain problems

can be improved significantly using multigrid techniques. A discussion of basic iterative methods is required before multigrid techniques can be presented.

B. BASIC ITERATIVE METHODS

Let **Au** = **f** denote the linear system in equation (5). The exact solution, **u**, to **Au=f** is unknown but the approximation **v** is known. There are two important measures of the quality of the approximation **v**. The first is the *error*, denoted by **e=u-v**. The size of the error can be measured by any vector norm such as $\|e\|_\infty = \max_{1 \le j \le N} |e_j|$. Since the error cannot be found if the exact solution is unknown, consider another important measure, the *residual*. The residual **r** is denoted by **r=f-Av**; it represents the amount by which the approximation fails to satisfy the equation. The residual can be measured by a vector norm in the same manner as the error. Rewrite **r=f-Av** as **Av=f-r** and subtract it from the equation **Au=f**. This results in the equation **Ae=r**, referred to as the *residual equation*. Multigrid methods employ iterative improvement, that is, the residual equation is solved (approximately) for **e** and a new approximation is computed to improve **v** in the next iteration using the definition **u=v+e**. That is, $\mathbf{v}^{(n+1)}=\mathbf{v}^{(n)}+\hat{\mathbf{e}}$, where $\hat{\mathbf{e}}$ is an approximation to **e**.

Consider the Jacobi relaxation method when applied to the model problem, rewritten as $-u_{j-1}+2u_j-u_{j+1}=h^2 f_j$, where $1 \le j \le N-1$, $u_0=u_N=0$ *and* $\sigma=0$. Let $\mathbf{v}^{(n)}$ be the current approximation and $\mathbf{v}^{(n+1)}$ be the updated approximation. The Jacobi method consists of solving the j^{th} equation for the j^{th} unknown u_j, holding all of the other unknowns fixed. When this has been done for all *N-1* equations, we

have performed one *sweep* of the Jacobi iteration. That is, for $j=1,2,\dots,N-1$, compute $v_j^{(1)} = \frac{1}{2}\left(v_{j-1}^{(0)} + v_{j+1}^{(0)} + h^2 f_j\right)$, $1 \le j \le N-1$. Ideally, these iteration sweeps are computed until convergence to the solution is attained.

Typically, after a few sweeps the convergence stalls, meaning that the answer will not improve with more iterations. This is because the error is made up of oscillatory and smooth components, and many iterative methods eliminate the oscillatory components of the error while having very little effect on the smooth components. This is known as the "smoothing property." The problem is now how to reduce the smooth components of the error that still exist. Multigrid methods evolved from attempts to solve this problem. The following sections address this problem in greater depth.

C. FOURIER MODES

A short discussion of Fourier modes is necessary to illustrate a key idea in multigrid methods. Consider the model problem $-u''(x)=0$. This is discretized and yields a simple linear homogenous system $\mathbf{Au} = \mathbf{0}$. The unique solution is $\mathbf{u=0}$ and the error in the approximate solution $\mathbf{v}$ is $-\mathbf{v}$. The linear system $\mathbf{Au=0}$ becomes $u_{j-1}+2u_j-u_{j+1}=0$, where $u_0=u_N=0$ and $1 \le j \le N-1$. Let the initial guess (and hence the initial error) be one the Fourier modes, that is, a vector $\mathbf{w}_k$, whose j^{th} component is $w_{kj} = \sin\left(\frac{jk\pi}{N}\right)$ when $0 \le j \le N$ and k is between 1 and $N-1$. The integer k is the *wavenumber* and it represents the number of half-sine waves which constitute $\mathbf{w}_k$ on the domain of the problem. Small values of k correspond to long

smooth waves and large values of k correspond to highly oscillatory waves. Figure 5 displays several Fourier modes on a grid with $N=12$. The k^{th} *mode* consists of k half sine waves and has wavelength $l=2/k$ (recall that the length of the interval is 1). The $k = N/2$ mode has a wavelength of $4h$ and the $k = N-1$ mode has a wavelength of $2h$. Waves with wavenumbers greater than N or wavelengths less than $2h$ cannot be represented on the grid. Through a phenomenon known as "aliasing", a wave with length less than $2h$ actually appears with a wavelength greater than $2h$. Wavenumbers where $1 \leq k \leq N/2$ are low frequency or smooth modes, and wavenumbers where $N/2 \leq k \leq N-1$ are high frequency or oscillatory modes.

For the cases $\mathbf{w}_2$ and $\mathbf{w}_{12}$, ten relaxation sweeps are performed on a grid with N=64 (see Figure 6). After ten relaxation sweeps, the high-frequency modulation on the long wave is nearly eliminated, but the smooth component remains. The key point of this discussion is that there is a fast rate of convergence as long as the error has high frequency components and convergence seems to stall when only low frequency components remain.

This supports the earlier observation that for oscillatory waves the error is damped very easily, but the smooth components remain essentially unchanged and so convergence stalls. The multigrid approach is to treat the smooth component of the error on a coarser grid where it will appear more oscillatory and can be damped so that fast convergence to the solution is possible.

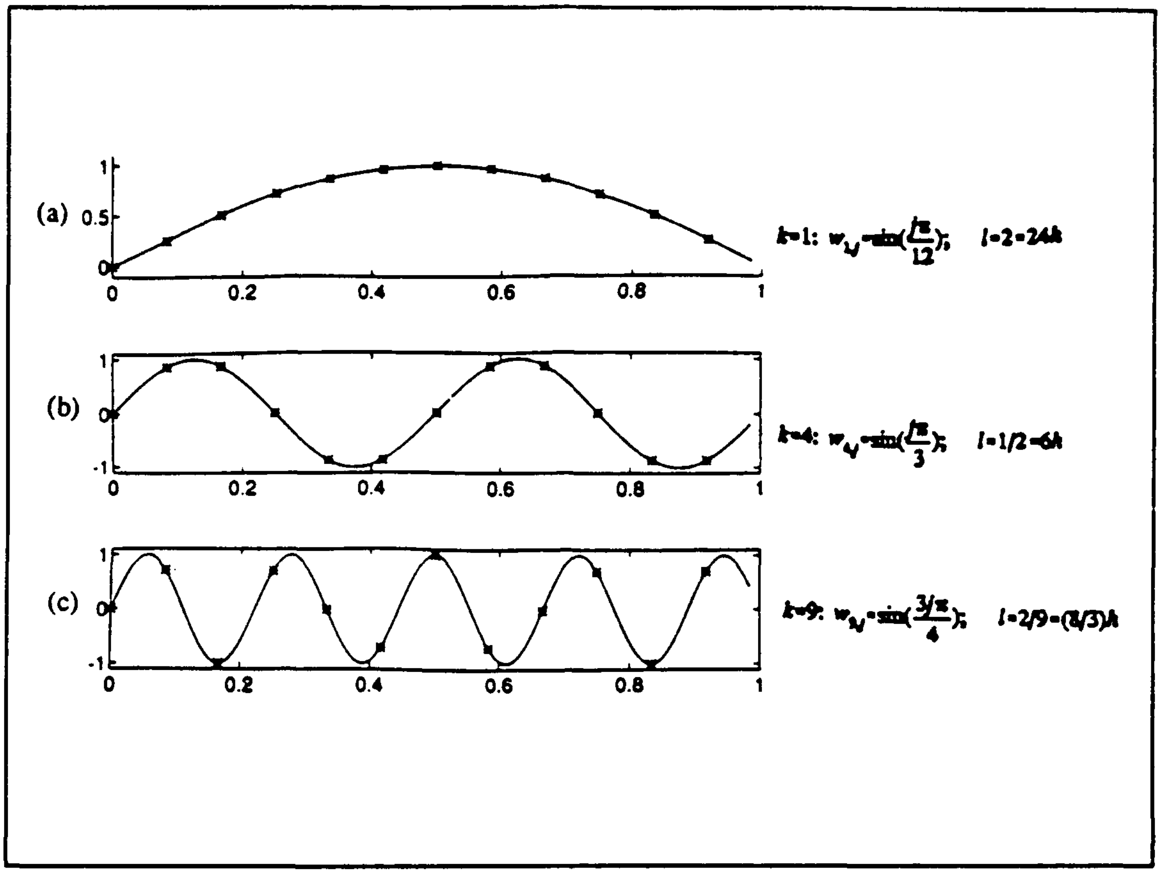

Figure 5. Graphs of the Fourier modes on a grid with $N=12$, for wavenumbers $k=1, 4, 9$.

D. COMPONENTS OF MULTIGRID

Suppose a relaxation scheme is applied until only the smooth error components remain. The smooth wave on the grid Ω^h looks more oscillatory on the grid Ω^{2h}. This is because the mode k is the same, but it is the k^{th} out of $N/2$, rather than k^{th} out of N. Also the grid points on Ω^{2h} are the even numbered points of Ω^h. Consider the k^{th} mode of Ω^h evaluated at the even-numbered grid points.

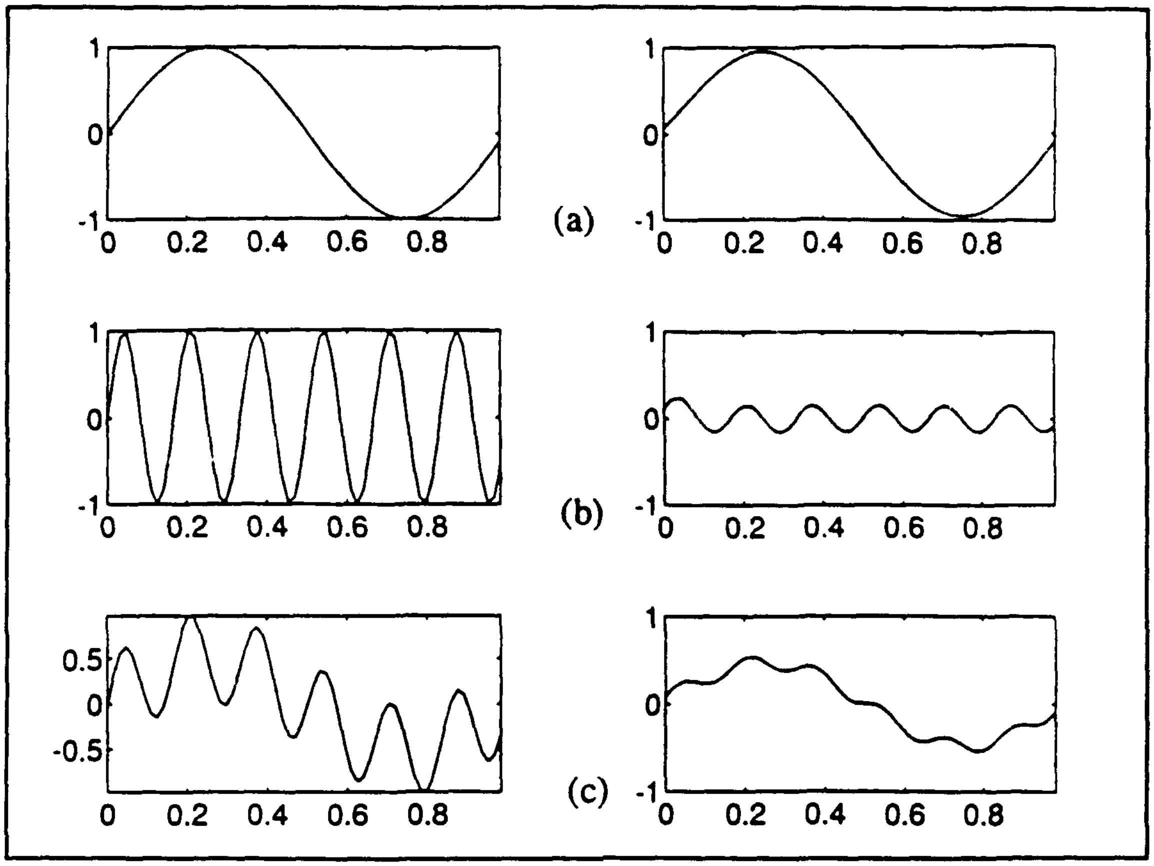

Figure 6. Initial guess (a) $\mathbf{w}_2$, (b) $\mathbf{w}_{12}$ and (c) combination of $\mathbf{w}_2$ and $\mathbf{w}_{12}$. The figures on the left show the approximation before iteration. The figure on the right is after ten iterations.

Then if $1 \leq k \leq N/2$, its components may be written as

$$w^h_{k,2j} = \sin\left(\frac{2jk\pi}{N}\right) = \sin\left(\frac{jk\pi}{N/2}\right) = w^{2h}_{k,j}, \quad 1 \leq k \leq \frac{N}{2}$$

One can say that, under restriction, the kth mode on Ω^h becomes the kth mode on Ω^{2h}. Therefore, transferring from Ω^h to Ω^{2h}, the k^{th} mode becomes more oscillatory. This is true for wavenumbers k, where $1 \leq k \leq N/2$ (see Figure 7).

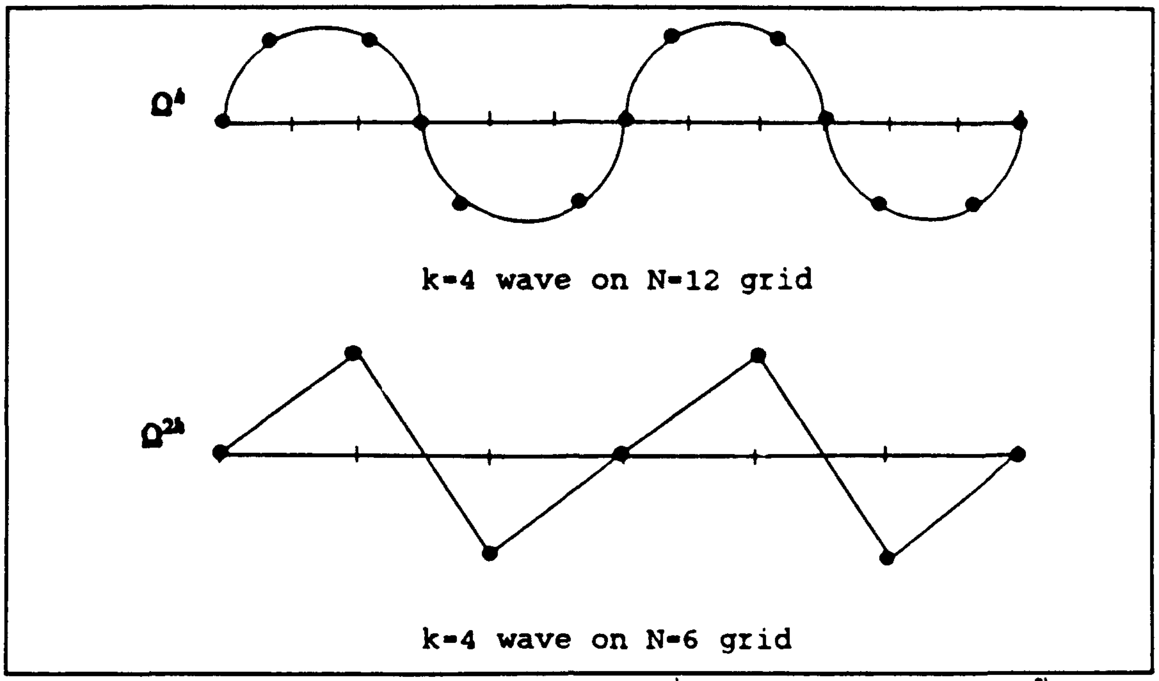

Figure 7. A wave with wavenumber k=4 on Ω^h(N=12) projected onto Ω^{2h}(N=6).

For wavenumbers k, where k>N/2, the oscillatory waves on Ω^h are aliased to become smooth on Ω^{2h}. This implies that the high frequency error must be damped *before* moving to the coarse grid. Two important features to note are:

1. Because the error is smooth after relaxation, it can be represented accurately on Ω^{2h}.

2. The smooth error on Ω^h appears more oscillatory on Ω^{2h} where it is more easily eliminated, since relaxation is cheaper.

This leads to the conclusion that an improvement to iterative methods would be to transfer the smooth error components to the coarse grid, where the error modes appear more oscillatory, and to then perform relaxation. Recall that there exists an equation for the error, namely **e=u-v**, satisfying **Ae=r=f-Av**. Therefore the residual equation can be used to relax directly on the error. Relaxing on **Au=f**

with an initial guess **v** is equivalent to relaxing on the residual equation **Ae=r** with initial guess **u-v**.

Nested iteration and *coarse grid correction* are two techniques that utilize the coarse grids. Coarse grid correction uses the coarse grid to improve the current approximation. Nested iteration uses the coarse grid to improve the initial guess.

An important question now arises: How does one move from the fine grid Ω^h to a coarser grid Ω^{2h}? The next two sections address this question of intergrid transfers.

1. Interpolation

The method used to transfer information from the coarse grid to the fine grid is *interpolation,* also referred to as *prolongation*. There are several interpolation methods to choose from, but only linear interpolation will be discussed here. I_{2h}^{h} is the linear interpolation operator. Consider only coarse to fine grid spacings with a ratio of 2:1. I_{2h}^{h} is a linear operator that transforms coarse grid vectors into fine grid vectors, that is $I_{2h}^{h}v^{2h}=v^{h}$. For the model problem this means

$$v_{2j}^{h}=v_{j}^{2h},\quad v_{2j+1}^{h}=\frac{1}{2}\left(v_{j}^{2h}+v_{j+1}^{2h}\right),\quad 0\le j\le\frac{N}{2}-1$$

Figure 8(a) illustrates this transformation.

2. Restriction

The method used to transfer information from the fine grid to the coarse grid is *restriction*. There are also several restriction methods available, but only full-weighting will be presented here. The restriction operator I_h^{2h} transforms fine grid vectors into coarse grid vectors, that is $I_h^{2h}\, v^h = v^{2h}$. For the model problem this means

$$v_{2j}^{h} = \frac{1}{4}\left(v_{j-1}^{h} + 2v_{j-1}^{h} + v_{2j+1}^{h}\right), \quad 1 \le j \le \frac{N}{2} - 1$$

This method takes a weighted average of neighboring fine grid points. Figure 8(b) illustrates this transformation.

3. Coarse Grid Correction

Coarse grid correction incorporates the use of the residual equation to relax on the error. This method requires relaxation on the fine grid until convergence stalls, then relaxes on the residual equation on the coarse grid to obtain an approximation to the error, then returns to the fine grid to correct the initial approximation. Coarse grid correction is a two grid process, and may be written as:

1. Relax on $A^h u^h = f^h$ to get v^h.
2. Restrict $r^{2h} = I_h^{2h}(f^h - A^h v^h)$.
3. Solve $A^{2h} e^{2h} = r^{2h}$ to get e^{2h}.

4. Interpolate and correct $v^h \leftarrow v^h + I_{2h}^h e^{2h}$ (Briggs, 1987).

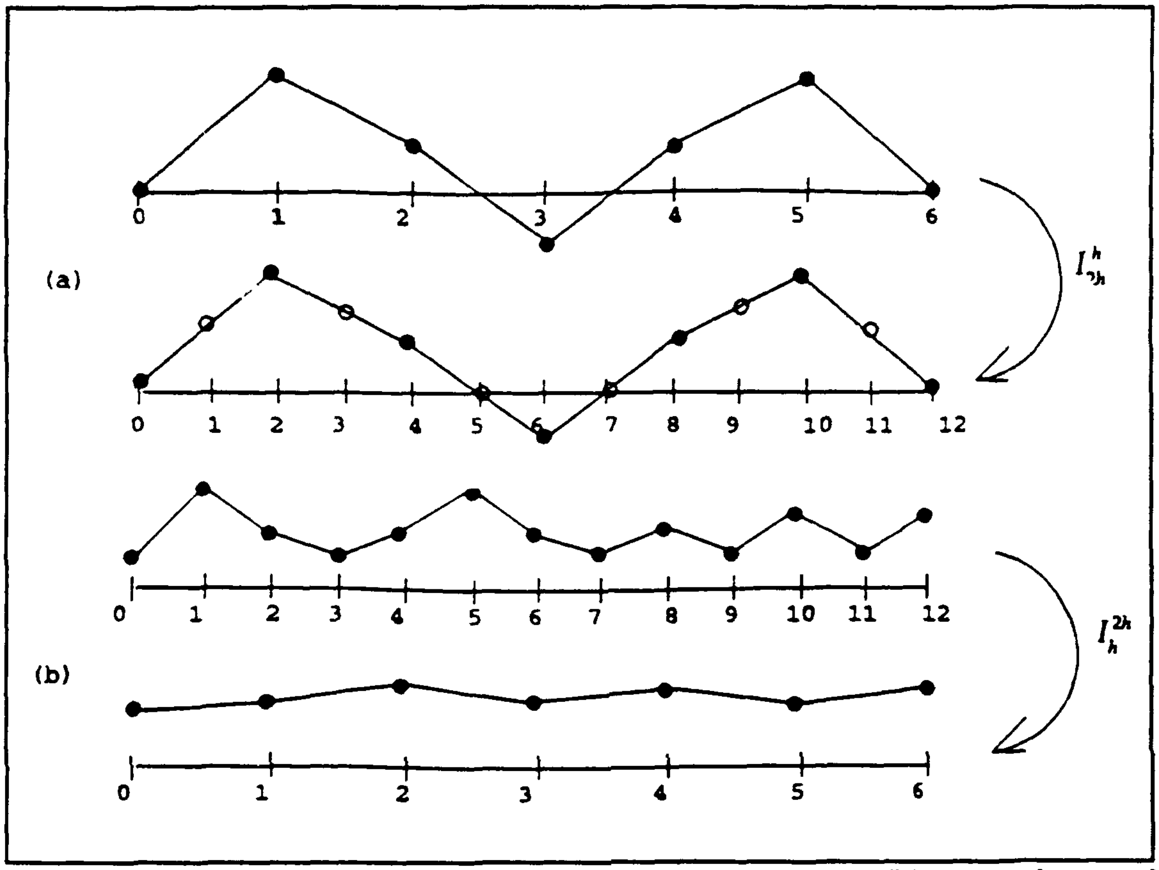

Figure 8. (a) Restriction of a fine grid vector to a coarser grid. (b) Interpolation of a coarse grid vector to a fine grid.

This leads to the question: How do we solve $A^{2h}e^{2h}=r^{2h}$ for the coarse grid scheme? The solution is to recursively apply coarse grid correction. This leads to the V-cycle.

4. The V-Cycle

The V-cycle algorithm applies the coarse grid correction scheme recursively until the coarsest grid is reached, and a direct solution to the residual equation is possible. It then works its way back to the finest grid. Note that as this correction scheme is applied recursively to solve $\mathbf{A}^{2h}\mathbf{e}^{2h}=\mathbf{r}^{2h}$ the notation can be simplified by letting $\mathbf{r}^{2h}=\mathbf{f}^{2h}$ and $\mathbf{e}^{2h}=\mathbf{v}^{2h}$. The coarse grid correction scheme applied recursively is presented next. Let Q denote the number of grids, where $Q>1$, and let $L=2^{Q-1}$ so that the coarsest grid has spacing Lh.

Relax on $\mathbf{A}^h\mathbf{u}^h=\mathbf{f}^h$ υ_1 times with initial guess $\mathbf{v}^h$.

Compute $\mathbf{f}^{2h}= I_h^{2h}\ \mathbf{r}^h$.

 Relax on $\mathbf{A}^{2h}\mathbf{u}^{2h}=\mathbf{f}^{2h}$ υ_1 times with initial guess $\mathbf{v}^{2h}=\mathbf{0}$.

 Compute $\mathbf{f}^{4h}= I_{2h}^{4h}\ \mathbf{r}^{2h}$.

  Relax on $\mathbf{A}^{4h}\mathbf{u}^{4h}=\mathbf{f}^{4h}$ υ_1 times with initial guess $\mathbf{v}^{4h}=\mathbf{0}$.

  Compute $\mathbf{f}^{8h}= I_{4h}^{8h}\ \mathbf{r}^{4h}$.

   ...

   Solve $\mathbf{A}^{Lh}\mathbf{u}^{Lh}=\mathbf{f}^{Lh}$.

   ...

  Correct $\mathbf{v}^{4h}\leftarrow\mathbf{v}^{4h}+ I_{8h}^{4h}\ \mathbf{v}^{8h}$.

  Relax on $\mathbf{A}^{4h}\mathbf{u}^{4h}=\mathbf{f}^{4h}$ υ_2 times with initial guess $\mathbf{v}^{4h}$.

 Correct $\mathbf{v}^{2h}\leftarrow\mathbf{v}^{2h} + I_{4h}^{2h}\ \mathbf{v}^{4h}$.

 Relax on $\mathbf{A}^{2h}\mathbf{u}^{2h}=\mathbf{f}^{2h}$ υ_2 times with initial guess $\mathbf{v}^{2h}$.

Correct $\mathbf{v}^h\leftarrow\mathbf{v}^h+ I_{2h}^{h}\ \mathbf{v}^{2h}$.

Relax on $\mathbf{A}^h\mathbf{u}^h=\mathbf{f}^h$ υ_2 with initial guess $\mathbf{v}^h$ (Briggs, 1987).

The V-cycle scheme, where $\mathbf{v}^h \leftarrow MV^h(\mathbf{v}^h,\mathbf{f}^h)$ may be defined recursively to give a compact description as follows:

1. Relax υ_1 times on $\mathbf{A}^h\mathbf{u}^h=\mathbf{f}^h$ with a given initial guess $\mathbf{v}^h$.
2. If Ω^h equals the coarsest grid, then go to 4.
   Else $\mathbf{f}^{2h} \leftarrow I_h^{2h}(\mathbf{f}^h-\mathbf{A}^h\mathbf{v}^h)$.
   $\mathbf{v}^{2h} \leftarrow \mathbf{0}$
   $\mathbf{v}^{2h} \leftarrow MV^{2h}(\mathbf{v}^{2h},\mathbf{f}^{2h})$.
3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h \mathbf{v}^{2h}$.
4. Relax υ_2 times on $\mathbf{A}^h\mathbf{u}^h=\mathbf{f}^h$ with initial guess $\mathbf{v}^h$ (Briggs, 1987).

The V-cycle is one of many multigrid cycling schemes. It is depicted in Figure 9(a).

Recall that the smoothing property is effective at eliminating the oscillatory components of the error while having little or no effect on the smooth components. We use coarse grid correction to eliminate the smooth error components. It is the combination of these techniques, relaxation for oscillatory error and coarse grid correction for smooth error, that makes multigrid such an effective method.

It stands to reason that if the initial guess is good, then the iterative scheme has less work to do in solving the problem. A useful technique is to first solve the problem on a coarse grid to obtain a good initial guess on the fine grid. This method is called *nested iteration*.

5. Nested Iteration

Nested iteration uses the coarsest grid to obtain an initial guess to the next finer grid and continues this process until the original grid is reached. Given the original problem **Au=f** posed on the coarsest grid, nested iteration proceeds as follows: Solve **Au=f** on the coarsest grid. Interpolate the solution to the next finer grid to obtain an initial guess. ...(repeat the process)...; solve **Au=f** on Ω^{4h} and interpolate to obtain an initial guess on Ω^{2h}; solve **Au=f** on Ω^{2h} and interpolate to obtain an initial guess on Ω^{h}. and solve **Au=f** on Ω^{h} to obtain the solution (Briggs, 1987). The algorithm that combines nested iteration with the V-cycle is known as the Full Multigrid (FMV) V-cycle.

6. The Full Multigrid V-Cycle

An efficient means of obtaining a good initial guess is to solve the problem first on a coarser grid and interpolate that solution for use as a fine-grid initial guess. This idea is nothing more than nested iteration. The next question to arise is: How can one solve the coarse-grid problem? One obvious choice is to use a V-cycle. This leads to the Full Multigrid (FMV) scheme. The FMV scheme where $\mathbf{v}^h \leftarrow \mathrm{FMV}^h(\mathbf{v}^h,\mathbf{f}^h)$ and $\mathbf{f}^h,\mathbf{f}^{2h},\ldots;\mathbf{v}^h,\mathbf{v}^{2h},\ldots$ are set equal to zero, is as follows:

1. If Ω^h is the coarsest grid, then go to step 3.
   Else $\mathbf{f}^{2h} \leftarrow I_h^{2h}(\mathbf{f}^h-\mathbf{A}^h\mathbf{v}^h)$
   $\mathbf{v}^{2h} \leftarrow \mathbf{0}$
   $\mathbf{v}^{2h} \leftarrow \mathrm{FMV}^{2h}(\mathbf{v}^{2h},\mathbf{f}^{2h})$.

2. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^{h}\mathbf{v}^{2h}$.

3. $v^h \leftarrow MV^h(v^h, f^h)$ υ_0 times (Briggs, 1987).

The FMV is depicted in Figure 9(b). Full multigrid is the culmination of the ideas presented in the preceding sections in this chapter. Multigrid combines each of the methods just described into an extremely efficient algorithm.

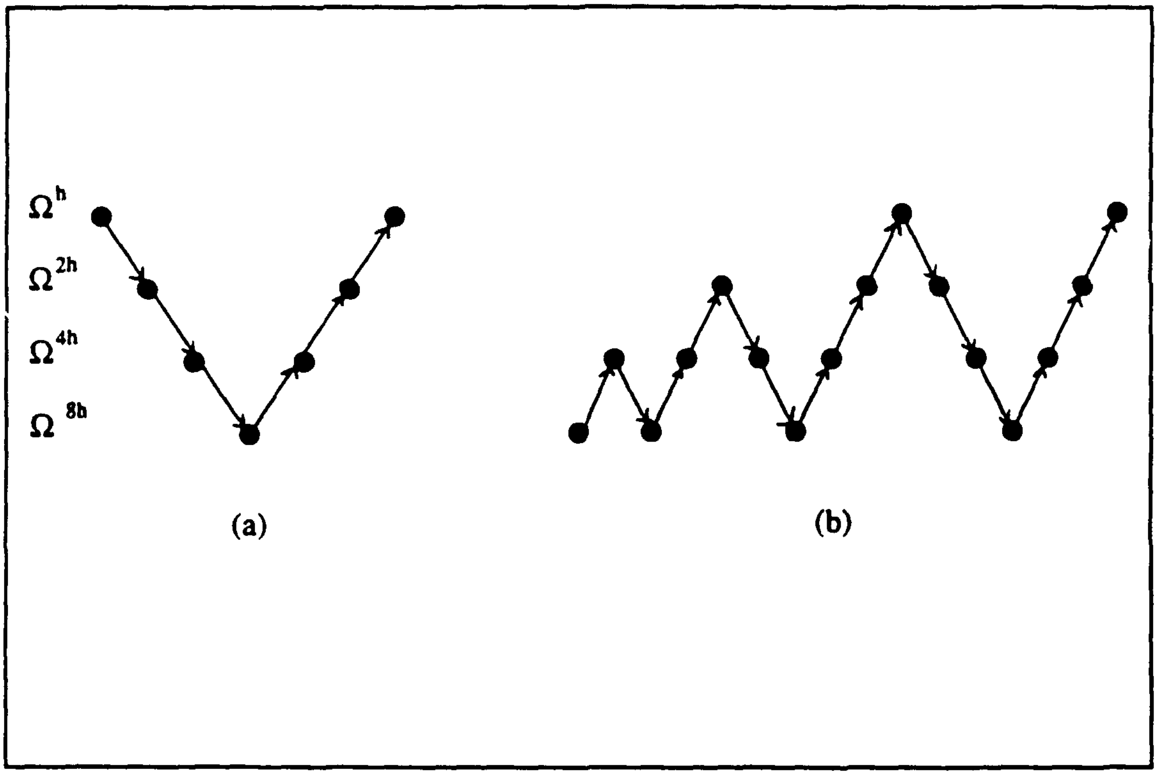

Figure 9. (a) The V-Cycle. (b) The FMV V-Cycle. The respective algorithms visit the various grids in the sequence indicated.

E. MULTIGRID METHODS APPLIED TO THE LONG TRANSPORTATION PROBLEM

One of the objectives of this thesis is to develop a multigrid algorithm to solve the long transportation problem. Analogies to the components of multigrid must be found. Recall that the key components in multigrid are interpolation,

restriction, and relaxation which are combined together to form the V-cycle. One major stumbling block is in drawing an analogy between some component of the transportation problem and the residual equation, which is key to multigrid solution methods. The following chapter presents the components of the algorithm developed during this thesis research.

IV. MULTIGRID COMPONENTS IN THE LONG TRANSPORTATION PROBLEM

A. BACKGROUND

As stated previously, a necessary assumption for the algorithm developed here is that the problem is posed on a regular grid. For instance, consider a problem with m supply nodes and n demand nodes posed on a regular m-dimensional grid. The dimension of the grid is determined by the number of supply nodes, hence m supply nodes implies an m-dimensional regular grid. The point identified by the three-tuple (i,j,k) denotes a particular demand node. The index i denotes the cost to ship to demand node (i,j,k) from supply node 1; the index j denotes the cost to ship from supply node 2 and the index k denotes the cost to ship from supply node 3. This can be written as $c_{1(ijk)}=i$, $c_{2(ijk)}=j$, and $c_{3(ijk)}=k$. Therefore, m supply nodes correspond to an m-tuple identifying each demand node. A three supply node problem with shipping costs mapped to a regular grid is displayed in Figure 10.

The method developed here takes a problem posed on Ω^h and transforms it into a problem on Ω^{2h} where it is easier to solve, it then interpolates the answer back to Ω^h. Three questions immediately come to mind. The first is: How is the problem transferred from Ω^h to Ω^{2h}? The second is: How is the problem solved on Ω^{2h}? And finally: How is the problem transferred from Ω^{2h} to back to Ω^h?

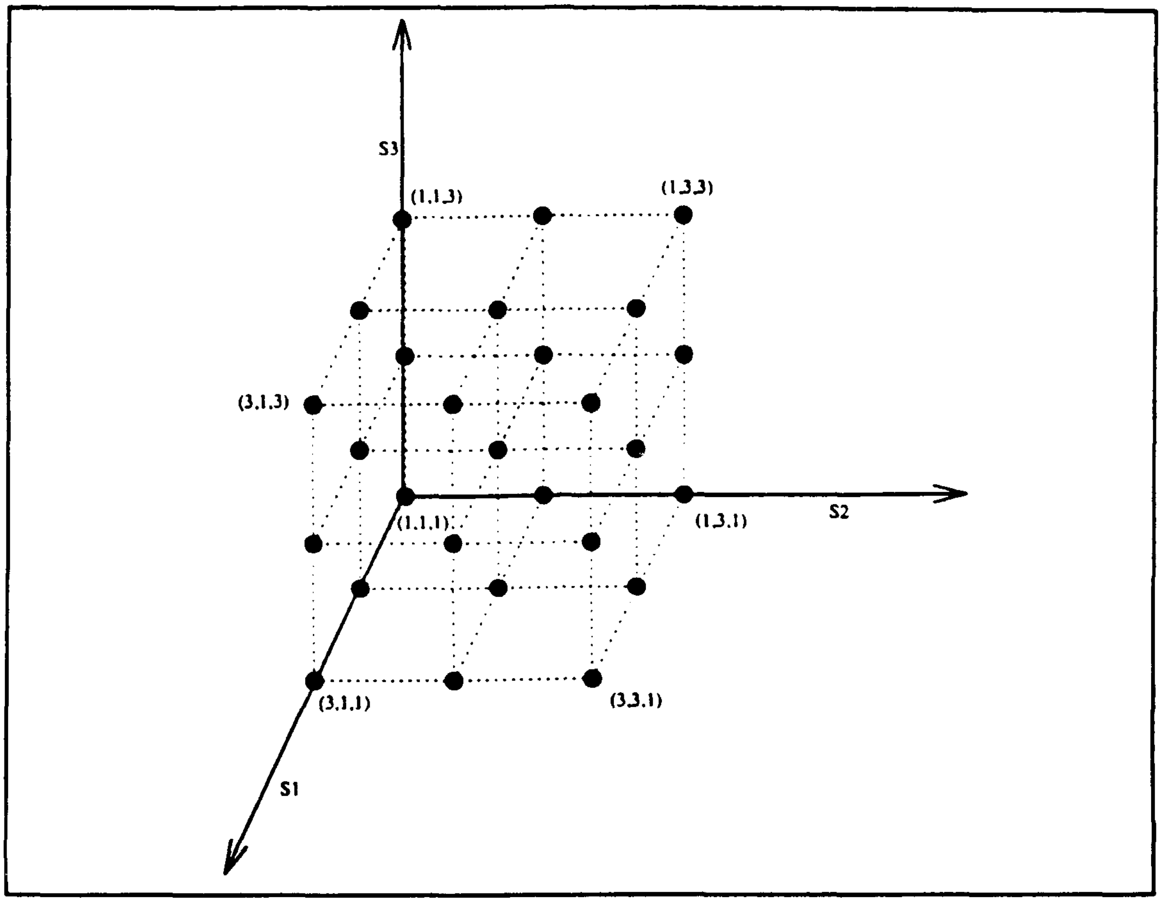

Figure 10. A Three Supply Node Problem Posed on a Regular Grid

These questions are all answered in the next four sections. But first an explanation of the grid and demand node relationship is necessary.

The finest grid in this algorithm contains $n=(2^t-1)^m$ demand nodes. The integer t represents the number of grid levels, m represents the dimension or the number of supply nodes. In what follows, we assume m=3; therefore, the most general problem considered will have size $3xn$. The *root of the r^{th}* grid is denoted by (2^r-1), where $r=1,...t$. Table I illustrates the relationship between the grid

number, the root of the grid and the number of demand nodes for up to five grids.

TABLE I. RELATIONSHIP BETWEEN GRIDS AND DEMAND NODES

Grid t	Dimension, m	Root, 2^t-1	Demand nodes $N=(2^t-1)^m$
1	3	1	1
2	3	3	27
3	3	7	343
4	3	15	3375
5	3	31	29,791

As shown above the number of grid levels is directly related to the number of demand nodes in the original problem. If four grids are chosen then $r=1,...,4$. The finest grid corresponds to $r=4$ and contains a *3x3375* transportation problem. The coarsest grid corresponds to $r=1$ and contains a *3x1* transportation problem. Recall that the spacing ratio between each grid in multigrid methods is *2:1*; the same convention is used here.

The analogies to restriction, interpolation, and relaxation developed during this research are presented next.

B. RESTRICTION

The restriction method developed transforms vectors containing fine grid demands into vectors containing coarse grid demands, that is, $\mathbf{d}^{2h}=I_h^{2h}\mathbf{d}^h$. The

coarsest grid transportation problem is the *3x1* problem. This problem is trivial to solve because the single demand node receives all the supply contained in each supply node.

To restrict from the fine grid Ω^h, with $(2^r\text{-}1)^3$ demand nodes to the next coarser grid, Ω^{2h} with $(2^{(r-1)}\text{-}1)^3$ demand nodes requires a method for aggregating the demand nodes to achieve a smaller problem. Let array *dh* contain the fine grid demand nodes and array *d2h* contain the next coarser grid demand nodes. The demand node *d2h(i,j,k)* on the r^{th} grid is made up of a weighted average of demand node *dh(2*i,2*j,2*k)* and the *26* demand nodes surrounding it on the $(r+1)^{st}$ grid. This is computed as follows:

$$
\begin{aligned}
&d2h(i,j,k) = dh(2*i,2*j,2*k) + (1/2)[dh(2*i-1,2*j,2*k) + dh(2*i,2*j-1,2*k)\\
&+dh(2*i,2*j,2*k-1) + dh(2*i+1,2*j,2*k) + dh(2*i,2*j+1,2*k)\\
&+dh(2*i,2*j,2*k+1)] + (1/4)[dh(2*i-1,2*j-1,2*k) + dh(2*i-1,2*j,2*k-1)\\
&+dh(2*i-1,2*j+1,2*k) + dh(2*i-1,2*j,2*k+1) + dh(2*i+1,2*j-1,2*k)\\
&+dh(2*i+1,2*j,2*k-1) + dh(2*i+1,2*j+1,2*k) + dh(2*i+1,2*j,2*k+1)\\
&+dh(2*i,2*j-1,2*k-1) + dh(2*i,2*j-1,2*k+1) + dh(2*i,2*j+1,2*k-1)\\
&+dh(2*i,2*j+1,2*k+1)] + (1/8)[dh(2*i-1,2*j-1,2*k-1) + dh(2*i-1,2*j+1,2*k-1)\\
&+dh(2*i-1,2*j+1,2*k+1) + dh(2*i-1,2*j-1,2*k+1) + dh(2*i+1,2*j+1,2*k-1)\\
&+dh(2*i+1,2*j-1,2*k+1) + dh(2*i+1,2*j+1,2*k+1) + dh(2*i+1,2*j-1,2*k-1)]
\end{aligned}
$$

The restriction method is summarized as: Restrict on Ω^h with $(2^t\text{-}1)^3$ demand nodes to the coarser grid Ω^{2h} with $(2^{(t-1)}\text{-}1)^3$ demand nodes.

C. INTERPOLATION

The interpolation step transforms vectors, containing the coarse grid flows, to vectors containing the fine grid flows. That is, $x^h = I_{2h}^{h} x^{2h}$. The interpolation uses a local solver to apportion the coarse grid flows. Each demand node on the coarse grid represents 27 demand nodes on the fine grid. Thus for each coarse-grid demand node we must apportion the coarse grid flows among the 27 fine grid demand nodes in the least expensive manner. This is done by treating the 3 coarse grid flows to a given coarse grid demand node as supplies, and solving a *3x27* transportation problem. This means the global "solution" on Ω^{2h} is interpolated to Ω^h by solving the one local *3x27* problem for each demand node *d2h(i,j,k)*, using the flow $x_{1(2i,2j,2k)}$ as supply s_1, the flow $x_{2(2i,2j,2k)}$ as supply s_2, and the flow $x_{3(2i,2j,2k)}$ as supply s_3. The global solution on Ω^h is the combination of all the local solutions. The *3x27* problems are what is referred to as the *local transportation problems*. There are $(2^{(r-1)}-1)^3$ local *3x27* transportation problems on the r^{th} grid. The local problems together comprise the *global problem* on the r^{th} grid (see Figure 11).

As an example, suppose that the demand at node *(3,1,2)* on a coarse grid is for 50 units of a particular commodity, and that the coarse grid solution indicates that it should receive 40 units from supply node two, 10 units from supply node three, and zero units from supply node one. These flows computed for demand node *(3,1,2)* now become supplies to the 27 associated demand nodes on the next finer grid. Hence, this local *3x27* problem has supplies s_1=0, s_2=40, and s_3=10.

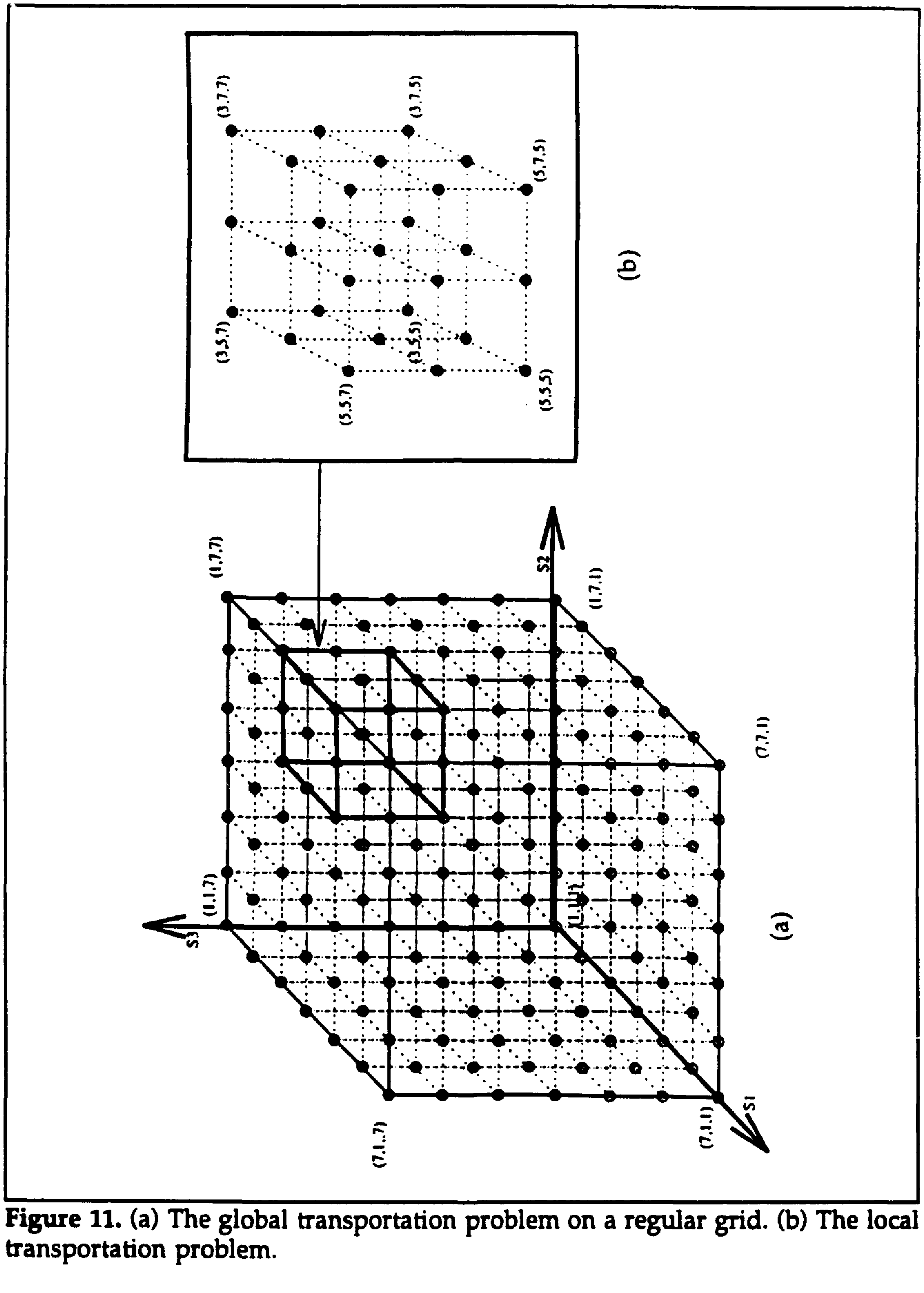

Figure 11. (a) The global transportation problem on a regular grid. (b) The local transportation problem.

The demands for the local problem on Ω^h must also be determined. Each fine grid demand node may be part of one, two, four, or eight local problems, depending on whether it lies in the interior, on a side, an edge, or a corner of a local problem. The total demand on a fine grid node must be apportioned equally among all the local problems that contain the node. This will ensure that local demand nodes that are included in two or more *3x27* problems do not receive more demand than required for the global demand node. The demand nodes on Ω^h receive some amount of demand from each of the demand nodes on Ω^h.

The interpolation process can be summarized as follows:

1. Determine local demands on Ω^h for each local *3x27* problem.
2. Solve each local *3x27* problem with the local solver, using the flows on the coarse-grid as supply values, and combine the local solutions.

D. LOCAL SOLVER

Given a solution on Ω^{2h}, it is interpolated by solving as many *3x27* problems as are necessary. The coarsest grid is a *3x1* problem, and only requires a single *3x27* local-solve for the first interpolation.

Because the problem is posed on a regular grid, there are thirteen distinct ways in which the three shipping costs for a demand node (denoted by *i,j*, and *k*) can be related. They are as follows:

1. $i=j=k$
2. $i=j,\ j>k$
3. $i=j,\ j<k$
4. $i=k,\ j>k$
5. $i=k,\ j<k$
6. $j=k,\ j>i$
7. $j=k,\ j<i$
8. $i<j<k$
9. $i<k<j$
10. $j<i<k$
11. $j<k<i$
12. $k<i<j$
13. $k<j<i$

The local solver exploits these patterns by choosing an order in which to fill the demands, using thirteen simple tests. Note that this is done *a priori,* in other words the list of ordered choices is hard-coded. This implies that none of this need be computed; it is free. If a sort routine were used the complexity would grow exponentially with the size of the problem; therefore, this method is much more efficient. The values of i,j, and k relative to one another determine the order in which the demand nodes will be supplied. The ordering is determined as follows:

1. For each demand node, determine the cost differential between the cheapest (minimum(i,j,k)) and the second cheapest supply node. This differential is denoted by Δ_1.

2. For each demand node, determine the cost differential between the second cheapest and the remaining supply node. The differential is denoted by Δ_2.

3. Form the string $\Delta_1\Delta_2$.

4. Lexicographically order the strings from largest to smallest. In the case of a tie the demand node with higher overall cost is placed first.

The supply nodes are then placed in order of preference for each demand node. The first choice supply node will be the cheapest supply node, the second choice will be the second cheapest and the third choice will be the remaining supply node. The local solver now fills each demand node in the predetermined order (the order given by the lexicographic ordering of the strings). For each demand node the method proceeds as follows:

1. Check the first choice supply node for adequate supply. If there is more supply than demand then the demand is filled entirely.
2. If there is more demand than supply and the supply does not equal zero, then fill the demand as much as possible from the first choice supply node.
3. If demand remains, check the second choice supply node. If possible, fill the remaining demand; if not, fill as much of the demand as possible from the second choice supply node.
4. If the demand is still not satisfied then fill the remaining demand from the third choice supply node.

This local solver fills each demand node requirement completely before repeating the process on the next demand node. The local solver provides an optimal solution to a 3x27 transportation problem.

Theorem 1. *Let* $\mathbf{x}$ *be the vector of flows determined by the algorithm above for the* 3×27 *problem. Then* $\mathbf{x}$ *is the optimal solution.*

Proof: Let $\mathbf{x}$ be the flow vector determined by the algorithm, $\mathbf{z} = \mathbf{cx}$, and let $\mathbf{z}^* = \mathbf{cx}^*$ be any optimal solution. Suppose that $\mathbf{x} \neq \mathbf{x}^*$. Without loss of generality

assume that $c_{1j} \le c_{2j} \le c_{3j}$. Let d_j be the first demand for which the flows **x** and **x***
differ, i.e., for all $1 \le i \le 3$ and $1 \le k \le j$ we have $x_{ik} = x^*_{ik}$. As a consequence of this and of the fact that the problem is balanced, for each $1 \le i \le 3$ we have $\sum_{k=j}^{27} (x_{ik} - x^*_{ik}) = 0$, from which we see that

$$x_{ij} - x^*_{ij} = \sum_{k=j+1}^{27} (x^*_{ik} - x_{ik}), \; i = 1,2,3 \tag{6}$$

Balance implies also that, for any choice of k, $\sum_{i=1}^{3} x_{ik} = d_k = \sum_{i=1}^{3} x^*_{ik}$, so for any i and k we have

$$x_{ik} - x^*_{ik} = \sum_{m=1}^{3} (x^*_{mk} - x_{mk}), \; m \ne i \tag{7}$$

If we let $\Delta = x_{1j} - x^*_{1j}$, then in particular (7) implies $\Delta = \sum_{i=2}^{3} (x^*_{ij} - x_{ij})$. Note that, since thealgorithm maximizes x_{1j}, then $\Delta > 0$. Now

$$\begin{aligned}
z^* &= z + z^* - z \\
&= z + cx^* - cx \\
&= z + c(x^* - x) \\
&= z + \sum_{i=1}^{3} \left[\sum_{k=1}^{j-1} c_{ik}(x^*_{ik} - x_{ik}) + c_{ij}(x^*_{ij} - x_{ij}) - \sum_{k=j+1}^{27} c_{ik}(x^*_{ik} - x_{ik}) \right]
\end{aligned} \tag{8}$$

By our choice of j, the first sum within the brackets in (8) vanishes, so now

$$
\begin{aligned}
z^* &= z + \sum_{i=1}^{3}\left[c_{ij}(x^*_{ij} - x_{ij}) + \sum_{k=j+1}^{27} c_{ik}(x^*_{ik} - x_{ik})\right] \\
&= z - c_{1j}\Delta + c_{2j}(x^*_{2j} - x_{2j}) + c_{3j}(x^*_{3j} - x_{3j}) \\
&\quad + \sum_{i=1}^{3}\sum_{k=j+1}^{27} c_{ik}(x^*_{ik} - x_{ik})
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
&= z - c_{1j}\Delta + c_{2j}(\Delta + (x_{3j} - x^*_{3j})) + c_{3j}(x^*_{3j} - x_{3j}) \\
&\quad + \sum_{i=1}^{3}\sum_{k=j+1}^{27} c_{ik}(x^*_{ik} - x_{ik}),
\end{aligned} \tag{10}
$$

where the equality of (9) and (10) follows from (7). Rearranging, we have

$$
z^* = z - (c_{1j} - c_{2j})\Delta + (c_{2j} - c_{3j})(x_{3j} - x^*_{3j}) + \sum_{i=1}^{3}\sum_{k=j+1}^{27} c_{ik}(x^*_{ik} - x_{ik}) \tag{11}
$$

$$
\begin{aligned}
&= z - (c_{1j} - c_{2j})\Delta + (c_{2j} - c_{3j})(x_{3j} - x^*_{3j}) + \sum_{k=j+1}^{27} c_{1k}(x^*_{1k} - x_{3k}) \\
&\quad + \sum_{k=j+1}^{27} c_{2k}(x_{1k} - x^*_{1k} + x_{3k} - x^*_{3k}) + \sum_{k=j+1}^{27} c_{3k}(x^*_{3k} - x_{3k})
\end{aligned} \tag{12}
$$

$$
\begin{aligned}
&= z - (c_{1j} - c_{2j})\Delta + (c_{2j} - c_{3j})(x_{3j} - x^*_{3j}) \\
&\quad + \sum_{k=j+1}^{27} (c_{1k} - c_{2k})(x^*_{1k} - x_{1k}) + \sum_{k=j+1}^{27} (c_{2k} - c_{3k})(x_{3k} - x^*_{3k}),
\end{aligned} \tag{13}
$$

where the equality of (11) and (12) follows from (6). Let $\delta_k = |c_{1k} - c_{2k}|$, and let $\gamma_k = |c_{2k} - c_{3k}|$. By hypothesis, $\delta_j = c_{2j} - c_{1j}$ and, by the action of the algorithm, if $k \neq j$, then $\delta_j \geq \delta_k$. For arbitrary values of k, the relationship between γ_j and γ_k is not clear, so let $\eta = \min\{-\gamma_j, -\gamma_k\}$. Then by rewriting (13) we have

$$z^* \geq z + \Delta\delta_j + (x_{3j} - x_{3j}^*)\eta + \sum_{k=j+1}^{27} (x_{1k}^* - x_{1k})(-\delta_j) + \sum_{k=j+1}^{27} (x_{3k} - x_{3k}^*)\eta$$

By (6) we have $\Delta = \sum_{k=j+1}^{27} (x_{1k}^*)$ and $(x_{3j}^* - x_{3j}) = \sum_{k=j+1}^{27} (x_{3k} - x_{3k}^*)$, whence

$$\begin{aligned} z^* &\geq z + \Delta(\delta_j - \delta_j) + \eta \sum_{k=j+1}^{27} ((x_{3k} - x_{3k}^*) - (x_{3k} - x_{3k}^*)) \\ &= z, \end{aligned}$$

and we conclude that $\mathbf{z} = \mathbf{cx}$ is optional. QED.

The next step is to try to improve the answer after each interpolation sweep before moving to the next finer grid. This is necessary because, even though the local *3x27* problems on the r^{th} grid are each solved to optimality, the global problem on the r^{th} grid probably won't be optimal. The reason for this is as follows.

Recall that an optimal solution to a transportation problem can always be found containing *n+m-1* arcs. Interpolating from the coarsest grid *(3x1)* requires solving a single *3x27* problem. In the solution to that problem, no more than 29 arcs have flow on them, hence at most two demand nodes receive supply from more than one supply node. As these flows are interpolated to the next finer grid this property may cause difficulty. To interpolate, the local solver proceeds to fill the demands of the $(r\text{-}1)^{st}$ grid local problem in the appropriate order. For example, suppose a demand node on the r^{th} grid is only supplied by supply node

one. A demand node in the local problem associated with this coarse grid node whose first choice supply node is two and second choice supply node is three can only receive supply from supply node one. This is clearly an optimal solution for the local *3x27* problem because there is only one supply node containing supply, but this is unlikely to be an optimal choice in the global problem. Hence, a relaxation scheme to improve the initial flow allocation before interpolating to the next finer grid is required. This relaxation scheme is described next.

E. RELAXATION

The relaxation method checks the flows computed during each interpolation sweep to determine if each demand node was filled by the cheapest possible supply node. If a demand node is not supplied by the cheapest supply node then the flow is diverted to a temporary demand node, and the flow to the original demand node is set equal to zero. Also, an equal amount of flow is added to a temporary supply node, such that if the misapportioned supply was from supply node i, the i^{th} temporary supply node receives the same amount of supply. If a demand node is supplied by the cheapest supply node then the demand in the corresponding temporary demand node is set equal to zero and no flow is added to the temporary supply node. Once all the demand nodes have been checked, the temporary demand and supply nodes become part of a "dummy" problem that has the same structure as the original problem but has only the supply that was misapportioned and has demand only at those nodes where the cheapest supplier

was not used. The local solver is applied to this global problem by solving as many local *3x27* problems as necessary. The relaxation scheme is summarized as follows:

1. If each demand node is supplied by the cheapest available supply node, go to step 3; else, go to step 2 .

2. Divert the flow to a temporary demand node and add the same amount of flow to a temporary supply node. Set the flow to the original demand node equal to zero. Go to step 4.

3. Set the corresponding temporary demand node equal to zero and do not add to the temporary supply node.

4. Compute the flows for the temporary supply and demand nodes.

5. Add the new flows to the existing solution. Repeat after each interpolation sweep.

This method improves the global flows because it only recomputes the flows for demand nodes not supplied by the cheapest supply node and it can now choose from all three supply nodes. In addition, it does not have any affect on the demand nodes already supplied by the cheapest supply node.

The four operations just described: restriction, interpolation, solve, and relaxation, are combined to form a V-cycle type of algorithm. The complete algorithm and the results of the tests performed are presented in the next chapter.

V. THE MODIFIED V-CYCLE

A. DESCRIPTION OF THE ALGORITHM

In this chapter, the components presented in Chapter IV are combined to form a V-cycle algorithm. Assume that there are $t>1$ grids. The grid spacing on the second coarsest grid, $\Omega^{(r/2)h}$, will equal qh where $q=2^{t-2}$. The coarsest grid consists of a single demand node, which necessarily receives all the supply. The algorithm can now be summarized as follows:

1. Restrict the original $3xn$ transportation problem posed on Ω^h to obtain the demands on all Ω^{2h}.
2. Solve the problem on Ω^{2h} and interpolate the solution to Ω^h.
3. Solve the problem on Ω^{2h} by restricting to Ω^{4h} then solving and interpolating back to Ω^{2h}.
4. Solve the problem on Ω^{4h} by restricting to Ω^{8h} then solving and interpolating back to Ω^{4h}.
5. This process continues until Ω^{rh} is reached where the problem is solved and interpolated back to $\Omega^{(r/2)h}$.
6. Compute the value of the objective function on Ω^h.

This process can be summarized recursively as

Ω^{h} Solve

Ω^{2h} Coarsen $\rightarrow \underset{\downarrow}{\underline{\text{Solve}}} \rightarrow$ Interpolate $\rightarrow$ Relax

Ω^{4h} Coarsen $\rightarrow \underset{\downarrow}{\underline{\text{Solve}}} \rightarrow$ Interpolate $\rightarrow$ Relax

... ... $\downarrow$

Ω^{rh} Coarsen $\rightarrow$ Solve $\rightarrow$ Interpolate $\rightarrow$ Relax

The net effect is a V-cycle depicted in Figure 12.

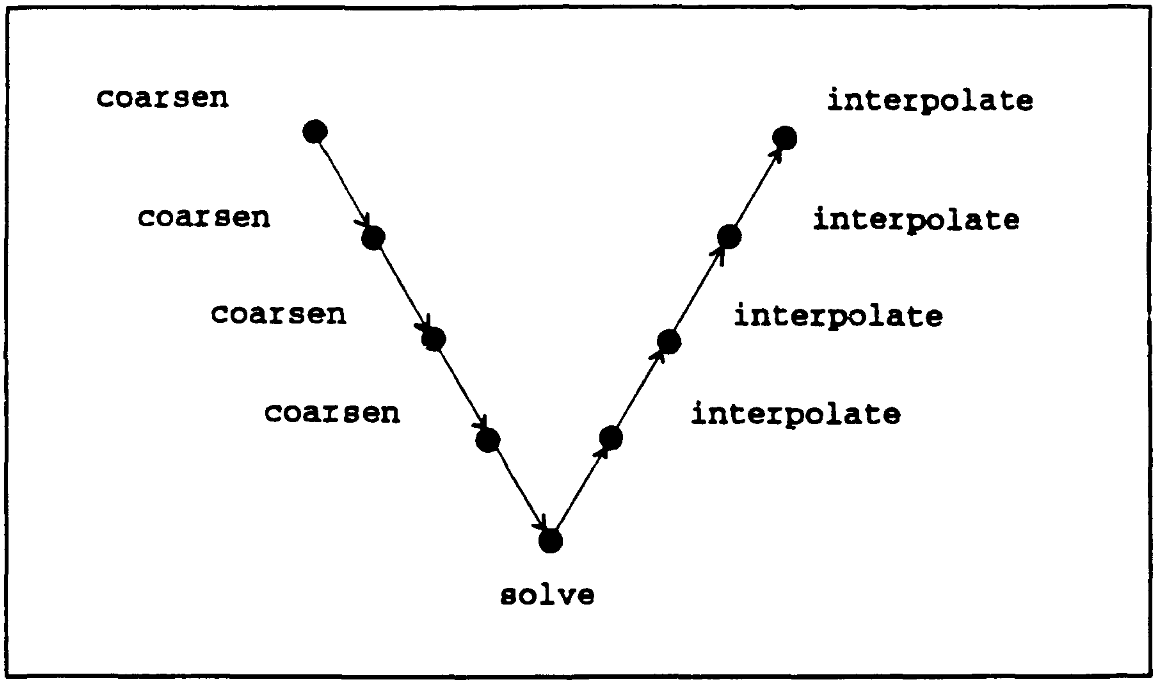

Figure 12. The V-Cycle Algorithm

This algorithm differs from the traditional multigrid V-cycle mainly in that there is no analogue to the residual equation. Thus the algorithm computes a solution of the coarse grid and interpolates it to the fine grid, rather than using the coarse grid to *correct* the fine grid solution. The relaxation scheme developed

here may provide the bridge to developing a FMV cycle, by giving a mechanism that can be used as a correction scheme. This will be discussed in Chapter VI.

This algorithm also has some key differences from aggregation methods. One difference is that aggregation/disaggregation is performed on only two levels; this algorithm is employed recursively on several levels. At first glance it may seem that this algorithm is increasing the work required to solve the problem, but this isn't necessarily so. Recall that on each level the problem gets smaller until it reaches a point where it is easy and computationally cheap to solve. This solution is then interpolated back up to the fine grid/original problem. This property is always exploited in multigrid so that a full multigrid V-cycle typically requires no more effort than two fine-grid relaxation sweeps. Another difference is that aggregation methods only use a subset of the data of the original problem to arrive at an answer; this algorithm uses all the information available. In other words, every demand node is individually filled during interpolation and then checked for optimality during relaxation. Finally, this algorithm uses the flows computed on the coarser grid to be supplies on the next finer grid, and there is no analogy to this in aggregation/disaggregation methods.

This algorithm is coded in FORTRAN and each component is written as a subroutine. The actual code consists of the driver code and four subroutines; Coarsen, Interp, Lsolve and Relax. The data structure used in this algorithm is similar to that used in traditional multigrid algorithms and it is outlined next.

B. DATA STRUCTURE

The arrays containing the supplies, demands and flows are the only permanent storage required. All other calculations are stored in temporary arrays which are overwritten on each iteration. The supply array requires three storage locations and the entries remain constant. The flow array must be three times longer than the demand array because flows from all three supply nodes to each demand node must be stored. Demands and flows are computed on each grid level; therefore, the demands and flows on each grid must be stored.

The data structure used stores the demands and flows, on each grid contiguously, in two arrays, one for demands and one for flows. Assume, for purposes of this argument, that the coarsest grid contains a single demand node. Thus a problem with five grids contains $(2^5-1)^3$ demand nodes on the finest grid, Ω^h; $(2^4-1)^3$ demand nodes on Ω^{2h}; $(2^3-1)^3$ demand nodes on Ω^{4h}; $(2^2-1)^3$ demand nodes on Ω^{8h}, and finally, a single demand node on Ω^{16h}. The total number of storage locations required for the demands is the sum of the storage required for the demands on each grid. The total number of storage locations required for the flows is three times that number. For $t>1$ grids, rh will represent the grid spacing on the coarsest grid, where $r=2^{t-1}$. In general, the finest grid requires

$$(2^t-1)^3+3(2^t-1)^3+3$$

storage locations. Since the supply storage requirements do not change from grid to grid, they are not included in the storage calculations. For ease of calculation let $N=2^t$ be the upper bound, since $2^t-1<2^t$. Thus Ω^h requires slightly fewer than

$4N^3$ storage locations for demands, flows, and supplies. Moving from any grid to the next coarser grid reduces the number of storage locations by a factor of $2^3=8$. Adding the storage requirements on each grid and using the sum of the resulting geometric series yields

$$Storage=4N^3(1+2^{-3}+2^{-6}+...+2^{-3t})<4N^3/(1-2^{-3})=(32/7)N^3$$

which is an upper bound on permanent storage locations.

C. COMPUTATIONAL COMPLEXITY

The driver code and the solve subroutine both have constant complexity, denoted by $O(1)$. For $t>1$ grids, rh represents the grid spacing on the coarsest grid, where $r=2^{t-1}$. The Coarsen subroutine has a constant number of calculations per grid point, denoted by J. These calculations are performed $(2^{(t-1)}-1)^3$ times on the finest grid, Ω^{2h}. These calculations are then performed $(2^{(t-2)}-1)^3$ on Ω^{h}, and so on until the grid preceding the coarsest grid is reached, $\Omega^{(r/2)h}$, where these operations are performed $(2^2-1)^3$ times. For ease of calculation, let $M=2^{t-1}$ be the upper bound, since $2^{(t-1)}-1<2^{(t-1)}$. Therefore, Ω^{h} requires JM^3 operations. Moving from any grid to the next coarser grid reduces the number of operations by a factor or $2^3=8$. This gives a total operation count $T_c = JM^3(1+2^{-3}+2^{-6}+...+2^{-3t})<JM^3/(1-2^{-3}) = (8/7)JM^3$.

A similar argument can be given for the Interp subroutine. Let K denote the number of constant calculations per grid point in Interp; therefore the total number of operations in Interp is given by $T_I=(8/7)KM^3$. For the Relax subroutine the same argument is used again, but in this case $N=2^t$ is the upper bound. This

is because the relaxation scheme is performed at most, $(2^{l})^3$ times, on any grid. Coarsen and Interp are performed at most $(2^{l-1})^3$ times on any grid. Let L denote the number of constant calculations required in Relax. Then the total number of calculations required in Relax is $T_R=(8/7)LN^3$. Considering the worst-case scenario, and the fact that $N>M$; the computational complexity of the entire algorithm can be expressed as $S(8/7)N^3$, where $S=J+K+L$. Therefore, this algorithm has $O(N^3)$ complexity.

D. PERFORMANCE

A separate program was written to generate random supply and demand node quantities, which were used to generate test problems. The supplies, demands and costs of the test problems were all integer valued. Approximately twenty sample transportation problems with random demand and supply quantities were solved using 3,4 and 5 grid levels. The average results on each grid are depicted in Table II. Table II shows that this algorithm comes consistently within 4% of optimality.

These results indicate that multigrid techniques can be successfully used to solve large scale long transportation problems posed in cost-space on regular grids. The solutions obtained are fairly close to optimality. While these results are very encouraging, there remain some difficult aspects of the algorithm to be addressed.

First, the algorithm produces real-valued demands and flows through the restriction and interpolation schemes. However, many real world problems of this type require integer output.

A second problem is that some flow is lost due to round-off error. Therefore, total supply does not equal total flow output. The flow loss increases with the size of the problem. For example, the *3x343* problem loses no measurable amount of flow. The *3x3375* problem loses *.0003%* of the total flow. And finally, the *3x29,791* problem loses *.0026%* of the total flow. This problem may disappear with improved restriction and interpolation schemes. These improved schemes must be able to preserve the integer values of the input to provide integer output.

A third problem stems from the fact that there are more than $m+n-1$ arcs having flow on them in the final solution, which is an indicator that the solution may not be optimal. For example, the *3x343* problem should contain, at most, *345* arcs with nonzero flow in the final solution, but there are actually *423* such arcs in the final solution. This implies that the underlying undirected graph of the solution to the transportation problem contains cycles. The interpolation scheme used in this algorithm is responsible for introducing these cycles. The cycles are a result of the coarse grid flows supplying only a portion of the demand for each demand node on the next finer grid. Thus each demand node on the next finer grid could ultimately be supplied by up to three different supply nodes. A search method to detect and eliminate cycles would be computationally expensive (Cavanaugh, 1992). However, revisions to the relaxation scheme used in this

algorithm could prove beneficial. Some suggestions are provided in the next chapter, as are conclusions, and recommendations for future research.

TABLE II. V-CYCLE-TYPE ALGORITHM

Number of Grids	3xn	Algorithm Time to Solve	% Over Optimal
2	27	0.001s	0
3	343	0.63s	3.8
4	3,375	8.0s	3.4
5	29,791	78.41s	4.0

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis shows that multigrid methods can be successfully applied to solution of large scale transportation problems when they are posed on a regular grid in cost space. The method developed here is not intended to replace any existing solution method. Instead the original goal of this thesis is to determine the applicability of the multilevel approach to solving the long transportation problem posed on a regular grid. The initial results are very encouraging, but more research is needed.

A significant contribution of this thesis is the identification of a relaxation scheme. The relaxation method developed is able to correct flows locally that are not supplied by the cheapest supply node. These corrections are made locally on the global grid on each grid level after the interpolation scheme. A method similar to this has not yet been developed for solving transportation problems using multigrid methods. In addition, this method could lead to a cycle detection scheme so that the solution would contain at most $m+n-1$ arcs in the final solution.

The relaxation scheme may also be the key to developing a full multigrid V-cycle. Call the misdirected flow, that is placed in the temporary array, the error.

Instead of solving several *3x27* local problems to correct the flow, solve one global problem. The global problem is of the form **Ae=r**. The question now arises: How does one solve **Ae=r**? The answer is to use the V-Cycle algorithm developed in this thesis. Recall that **u** can be replaced by **e** and **f** can be replaced by **r** in the equation **Au=f** to arrive at an equation of the same form, **Ae=r**. Suppose there exists an initial guess on the coarse grid, interpolate the flow to the next level and find the error using the relaxation scheme. Repeat until the fine grid answer is reached. This algorithm results in a Full Multigrid V-cycle algorithm depicted in Figure 9(b).

The problem solved in this thesis is deliberately artificial. Further advancements in this area of research require a solution method that is applicable to more general problems. Specifically, a solver that can handle more than three supply nodes must be developed.

B. AREAS FOR FUTURE RESEARCH

The results of this thesis indicate that further research in this area can be productive. There are several avenues for future research.

This algorithm solves a very specific problem that is posed on a three-dimensional grid. Generalization to an m-dimensional grid would logically follow. Research into developing a method to map an arbitrary transportation problem to a regular grid and then map the solution back to the original problem domain is needed.

Another area of research that is needed for the algorithm developed is to find a different method of restriction that will not introduce real-valued demands or flows into the problem. One such method could be analogous to the *finite volume* element discretization used in modern multigrid methods (McCormick, 1989). A new interpolation method that also preserves integer flows is necessary. These are the two most urgent areas of follow-on research aimed at correcting the problem of lost flow in the output. As stated earlier, most users require integer flows and costs.

A third area of research is to devise a more efficient relaxation scheme, or perhaps to develop a new relaxation, based on detecting and eliminating cycles (Cavanaugh, 1992). The scheme developed here already contains the necessary information, such as the flow for each demand node and the supplier. An efficient means of using this data to find cycles must be determined. A test similar to the existing one that checks for optimality could also check for multiple supply nodes to a single demand node and then redirect the flow.

Lastly and most importantly, this thesis provides a direction to begin to implement a true Full Multigrid V-cycle algorithm which has not been done before.

APPENDIX

```
C      THIS CODE WAS WRITTEN BY ANNETTE P. CORNETT
C                                 1606 KICKAPOO RD.
C                                 PUEBLO, CO 81001
C
C      THIS PROGRAM WILL SOLVE A LONG TRANSPORTATION PROBLEM ON A REGULAR
C      GRID IN COST SPACE.THE LARGEST SIZE PROBLEM SOLVED BY THIS PROGRAM
C      IS A 3X29,071 TRANSPORTATION PROBLEM.  THE PARAMETER, NGRIDS MUST
C      BE CHANGED BEFORE THE PROGRAM IS RUN.  THE ARRAY FLOW CONTAINS THE
C      FLOWS ON EACH GRID.  IT IS INITIALIZED TO ZERO IN THE BEGINNING.
C      THE ARRAY DEMANDS CONTAINS THE QUANTITY OF DEMAND FOR EACH DEMAND
C      NODE ON EACH GRID.  THE ARRAY SUPPLY CONTAINS THE INITIAL SUPPLY
C      FOR THE FINE GRID PROBLEM. THIS PROGRAM CONTAINS THREE
C      SUBROUTINES, THEY ARE: COARSEN, INTERP, LSOLVE AND RELAX.  A BRIEF
C      EXPLANATION FOR EACH PRECEDES THE ROUTINE.

       PROGRAM TRANS11
       INTEGER DIM
       PARAMETER (NGRIDS = 3,DIM = 3)
       REAL FLOW(90000),DEMANDS(30000),SUPPLY(DIM),XHTEMP(DIM,63,63,
      +63),DHTEMP(63,63,63),STEMP(DIM)
       INTEGER NROOT(NGRIDS),IDX(NGRIDS),ED(NGRIDS),NPTS(NGRIDS),L,COUNT
       COUNT =0
       ITEMP = 1
       JTEMP = 1
       KTEMP = 1
       DO 1 I=1,3
 1     STEMP(I)=0

C      SET UP INDEX SCHEME FOR ARRAYS DEMANDS AND FLOWS

       DO 5 I = 1,NGRIDS
          NROOT(I) = 2**I-1
          NPTS(I) = (2**I-1)**DIM
          IDX(I) = 1
          ED(I) = 1
  5    CONTINUE
       DO 10 J = 2,NGRIDS
          IDX(J) = IDX(J-1) + NPTS(J-1)
          ED(J) = IDX(J) + NPTS(J) -1
 10    CONTINUE
       K = IDX(NGRIDS) + NPTS(NGRIDS) - 1
       DO 15 I = 1,K
          DEMANDS(I) = 0
 15    CONTINUE
       DO 20 I = 1,3*K
          FLOW(I) = 0
 20    CONTINUE
       OPEN(UNIT = 6,FILE = 'FINE DATA D',STATUS = 'OLD')
       READ(6,22)(SUPPLY(I),I=1,DIM)
 22    FORMAT(F12.0)
       READ(6,22)(DEMANDS(I),I=IDX(NGRIDS),ED(NGRIDS))
       DO 25 I = NGRIDS,3,-1
```

```
25   CALL COARSEN(DEMANDS(IDX(I)),DEMANDS(IDX(I-1)),NROOT(I),NROOT(I-1)
    +)
      I=2
      CALL LSOLVE(FLOW(3*IDX(I)-2),DEMANDS(IDX(I)),SUPPLY,DIM,
    +ITEMP,JTEMP,KTEMP)
      DO 30 I = 2,NGRIDS-1
         DO 31 K = 1,NROOT(I)
            DO 31 J = 1,NROOT(I)
               DO 31 II = 1,NROOT(I)
                  DO 31 L = 1,DIM
                     XHTEMP(L,2*II,2*J,2*K) = 0
                    IF((K.LE.3).AND.(J.LE.3).AND.(II.LE.3).AND.(L.EQ.1)
    +)THEN
                        DHTEMP(II,J,K) = 0
                     ENDIF
31    CONTINUE
           CALL INTERP(DEMANDS(IDX(I+1)),DHTEMP,XHTEMP,STEMP,
    +FLOW(3*IDX(I)-2),FLOW(3*IDX(I+1)-2),NROOT(I),NROOT(I+1),DIM)
30         CALL RELAX(STEMP,FLOW(3*IDX(I+1)-2),NROOT(I),NROOT(I+1),DIM)

      END

C     THE SUBROUTINE COARSEN WILL RESTRICT THE FINE GRID DEMAND NODES TO
C      THE NEXT COARSER GRID BY USING A WEIGHTED AVERAGE. THE ARRAY
C      DH CONTAINS THE DEMAND FOR EACH FINE GRID DEMAND NODE.  THE
C      ARRAY D2H CONTAINS THE DEMAND FOR EACH FINE GRID DEMAND  NODE.
C      R IS THE ROOT OF THE FINE GRID, RC IS THE ROOT OF THE
C      COARSE GRID.

      SUBROUTINE COARSEN(DH,D2H,R,RC)
      INTEGER R,RC,SUM
      REAL IM,JM,KM,IP,JP,KP,IMJM,IMJP,IMKM,IMKP,JMKM,JMKP,IPJP,IPJM,
    +IPKP,IPKM,JPKP,JPKM,IMJMKM,IMJPKM,IMJMKP,IMJPKP,IPJPKP,IPJMKM,
    +IPJPKM,IPJMKP,DH(R,R,R),D2H(RC,RC,RC)

      SUM = 0
C     COMPUTE THE WEIGHTED AVERAGE

      DO 10 K = 1,RC
         DO 10 J = 1, RC
            DO 10 I = 1,RC
               D2H(I,J,K) = D2H(I,J,K) + DH(2*I,2*J,2*K)

               IM = .5
               IP = .5
               JM = .5
               JP = .5
               KM = .5
               KP = .5
               IMJM = .25
               IMJP = .25
               IPJM = .25
               IPJP = .25
               IMKM = .25
               IMKP = .25
               IPKM = .25
               IPKP = .25
               JMKM = .25
               JMKP = .25
               JPKM = .25
```

```
            JPKP = .25
            IMJMKM = .125
            IMJPKM = .125
            IMJMKP = .125
            IMJPKP = .125
            IPJMKM = .125
            IPJMKP = .125
            IPJPKM = .125
            IPJPKP = .125

C           THE IF STATEMENTS CHECK FOR EDGE AND CORNER NODES

            IF (I .EQ. 1) THEN
               IM = 1
               IMJM = .5
               IMJP = .5
               IMKM = .5
               IMKP = .5
               IMJPKM = .25
               IMJMKP = .25
               IMJPKP = .25
               IMJMKM = .25

            ELSEIF(I .EQ.RC) THEN
               IP = 1
               IPJM = .5
               IPJP = .5
               IPKM = .5
               IPKP = .5
               IPJPKM = .25
               IPJMKP = .25
               IPJPKP = .25
               IPJMKM = .25
            ENDIF
            IF (J .EQ. 1) THEN
               JM = 1
               IMJM = .5
               IPJM = .5
               JMKM = .5
               JMKP = .5
               IMJMKM = .25
               IPJMKP = .25
               IMJMKP = .25
               IPJMKM = .25
            ELSEIF (J .EQ.RC) THEN
               JP = 1
               IMJP = .5
               IPJP = .5
               JPKM = .5
               JPKP = .5
               IMJPKM = .25
               IMJPKP = .25
               IPJPKP = .25
               IPJPKM = .25
            ENDIF
            IF (K .EQ. 1)  THEN
               KM = 1
               IMKM = .5
               IPKM = .5
               JMKM = .5
```

```
            JPKM = .5
            IMJPKM = .25
            IMJMKM = .25
            IPJPKM = .25
            IPJMKM = .25
         ELSEIF (K .EQ.RC) THEN
            KP = 1
            IMKP = .5
            IPKP = .5
            JMKP = .5
            JPKP = .5
            IMJPKP = .25
            IMJMKP = .25
            IPJPKP = .25
            IPJMKP = .25
         ENDIF

         D2H(I,J,K) = D2H(I,J,K) + IM*DH(2*I-1,2*J,2*K)
     +   + IP*DH(2*I+1,2*J,2*K) + JM*DH(2*I,2*J-1,2*K)
     +   + JP*DH(2*I,2*J+1,2*K)+ KM*DH(2*I,2*J,2*K-1)
     +   + KP*DH(2*I,2*J,2*K+1)

         IF ((I .EQ.1) .AND. (J .EQ. 1)) THEN
            IMJM = 1
            IMJMKP = .5
            IMJMKM = .5
         ELSEIF ((I .EQ.1) .AND. (J .EQ.RC)) THEN
            IMJP = 1
            IMJPKP = .5
            IMJPKM = .5
         ENDIF
         IF ((I .EQ.RC) .AND. (J .EQ. 1)) THEN
            IPJM = 1
            IPJMKP = .5
            IPJMKM = .5
         ELSEIF ((I .EQ.RC) .AND. (J .EQ.RC)) THEN
            IPJP = 1
            IPJPKP = .5
            IPJPKM = .5
         ENDIF
         IF ((I .EQ.1) .AND. (K .EQ. 1)) THEN
            IMKM = 1
            IMJMKM = .5
            IMJPKM = .5
         ELSEIF ((I .EQ.1) .AND. (K .EQ.RC)) THEN
            IMKP = 1
            IMJMKP = .5
            IMJPKP = .5
         ENDIF
         IF ((I .EQ.RC) .AND. (K .EQ. 1)) THEN
            IPKM = 1
            IPJMKM = .5
            IPJPKM = .5
         ELSEIF ((I .EQ.RC) .AND. (K .EQ.RC)) THEN
            IPKP = 1
            IPJMKP = .5
            IPJPKP = .5
         ENDIF
         IF ((J .EQ. 1) .AND. (K .EQ. 1)) THEN
            JMKM = 1
```

```
            IMJMKM = .5
            IPJMKM = .5
         ELSEIF ((J .EQ. 1) .AND. (K .EQ.RC)) THEN
            JMKP = 1
            IMJMKP = .5
            IPJMKP = .5
         ENDIF
         IF ((J .EQ.RC) .AND. (K .EQ. 1)) THEN
            JPKM = 1
            IMJPKM = .5
            IPJPKM = .5
         ELSEIF ((J .EQ.RC) .AND. (K .EQ.RC)) THEN
            JPKP = 1
            IMJPKP = .5
            IPJPKP = .5
         ENDIF
         D2H(I,J,K) = D2H(I,J,K) + IMJM*DH(2*I-1,2*J-1,2*K)
     +   + IPJM*DH(2*I+1,2*J-1,2*K) + IMJP*DH(2*I-1,2*J+1,2*K)
     +   + IPJP*DH(2*I+1,2*J+1,2*K) + IMKM*DH(2*I-1,2*J,2*K-1)
     +   + IMKP*DH(2*I-1,2*J,2*K+1) + IPKM*DH(2*I+1,2*J,2*K-1)
     +   + IPKP*DH(2*I+1,2*J,2*K+1) + JMKM*DH(2*I,2*J-1,2*K-1)
     +   + JMKP*DH(2*I,2*J-1,2*K+1) + JPKM*DH(2*I,2*J+1,2*K-1)
     +   + JPKP*DH(2*I,2*J+1,2*K+1)


         IF ((I.EQ.1).AND.(J.EQ.1).AND.(K.EQ.1)) THEN
            IMJMKM = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.1).AND.(K.EQ.RC)) THEN
            IMJMKP = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.RC).AND.(K.EQ.1)) THEN
            IMJPKM = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.RC).AND.(K.EQ.RC)) THEN
            IMJPKP = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.1).AND.(K.EQ.1)) THEN
            IPJMKM = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.RC).AND.(K.EQ.1)) THEN
            IPJPKM = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.RC).AND.(K.EQ.RC)) THEN
            IPJPKP = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.1).AND.(K.EQ.RC)) THEN
            IPJMKP = 1
         ENDIF

        D2H(I,J,K) = D2H(I,J,K) + IMJMKM*DH(2*I-1,2*J-1,2*K-1)
     +  +IPJMKM*DH(2*I+1,2*J-1,2*K-1)+IPJPKM*DH(2*I+1,2*J+1,2*K-1)
     +  +IPJMKP*DH(2*I+1,2*J-1,2*K+1)+IMJPKM*DH(2*I-1,2*J+1,2*K-1)
     +  +IMJMKP*DH(2*I-1,2*J-1,2*K+1)+IMJPKP*DH(2*I-1,2*J+1,2*K+1)
     +   +IPJPKP*DH(2*I+1,2*J+1,2*K+1)

 10   CONTINUE

      RETURN
      END

C     THE SUBROUTINE LSOLVE SOLVES THE LOCAL 3X27 TRANSPORTATION PROBLEM
C     THE ARRAY X2H CONTAINS THE FLOWS FOR EACH COARSE GRID DEMAND NODE.
C     THE ARRAY D2H CONTAINS THE DEMANDS FOR EACH LOCAL TRANSPORTATION
C     PROBLEM DEMAND NODE.  THE DEMAND NODES ARE FILLED IN LEXICOGRAPHIC
```

```
C      DEPENDING ON THE RELATIONSHIP BETWEEN THE COSTS FROM EACH SUPPLY
C      NODE. THE ARRAY D CONTAINS THE FIRST CHOICE SUPPLY NODE FOR EACH
C      DEMAND NODE.  THE ARRAY DP CONTAINS THE SECOND CHOICE SUPPLY NODE
C      FOR EACH DEMAND NODE.  THE ARRAY DPP CONTAINS THE THIRD CHOICE
C      SUPPLY NODE FOR EACH DEMAND NODE.

       SUBROUTINE LSOLVE (X2H,D2H,S,DIM,II,JJ,KK)
       INTEGER DIM,P,D(27),DP(27),DPP(27)
       REAL D2H(3,3,3), X2H(DIM,DIM,DIM,DIM),S(DIM), X(3,27),DEMAND(27),
      +DTEMP(27)

*     INITIALIZE ARRAYS
       DO 1 I = 1,27
          DTEMP(I) = 0
          D(I) = 0
          DP(I) = 0
          DPP(I) = 0
          DEMAND(I) = 0
 1     CONTINUE
       P = 1
       DO 5 L = 1,DIM
          DO 5 K = 1,3
             DO 5 J = 1,3
                DO 5 I = 1,3
                  X2H(L,I,J,K) = 0
                  IF (L .EQ. DIM) THEN
                     DTEMP(P) = D2H(I,J,K)
                     P = P + 1
                  ENDIF
5      CONTINUE
       DO 6 I = 1,DIM
          DO 6 J = 1,27
             X(I,J) = 0
 6     CONTINUE


*    REORDER DEMAND NODE ARRAY

       IF((II.EQ.JJ).AND.(JJ.EQ.KK)) THEN

          DEMAND(1) = DTEMP(9)
          DEMAND(2) = DTEMP(21)
          DEMAND(3) = DTEMP(25)
          DEMAND(4) = DTEMP(6)
          DEMAND(5) = DTEMP(8)
          DEMAND(6) = DTEMP(12)
          DEMAND(7) = DTEMP(16)
          DEMAND(8) = DTEMP(20)
          DEMAND(9) = DTEMP(22)
          DEMAND(10) = DTEMP(18)
          DEMAND(11) = DTEMP(24)
          DEMAND(12) = DTEMP(26)
          DEMAND(13) = DTEMP(5)
          DEMAND(14) = DTEMP(11)
          DEMAND(15) = DTEMP(13)
          DEMAND(16) = DTEMP(3)
          DEMAND(17) = DTEMP(7)
          DEMAND(18) = DTEMP(19)
          DEMAND(19) = DTEMP(15)
          DEMAND(20) = DTEMP(17)
```

```
DEMAND(21) = DTEMP(23)
DEMAND(22) = DTEMP(2)
DEMAND(23) = DTEMP(4)
DEMAND(24) = DTEMP(10)
DEMAND(25) = DTEMP(1)
DEMAND(26) = DTEMP(14)
DEMAND(27) = DTEMP(27)
D(1) = 3
DP(1) = 1
DPP(1) = 2
D(2) = 2
DP(2) = 1
DPP(2) = 3
D(3) = 1
DP(3) = 3
DPP(3) = 2
D(4) = 3
DP(4) = 2
DPP(4) = 1
D(5) = 3
DP(5) = 1
DPP(5) = 2
D(6) = 2
DP(6) = 3
DPP(6) = 1
D(7) = 1
DP(7) = 3
DPP(7) = 2
D(8) = 2
DP(8) = 1
DPP(8) = 3
D(9) = 1
DP(9) = 2
DPP(9) = 3
D(10) = 3
DP(10) = 2
DPP(10) = 1
D(11) = 2
DP(11) = 1
DPP(11) = 3
D(12) = 1
DP(12) = 3
DPP(12) = 2
D(13) = 3
DP(13) = 2
DPP(13) = 1
D(14) = 2
DP(14) = 1
DPP(14) = 3
D(15) = 1
DP(15) = 2
DPP(15) = 3
D(16) = 3
DP(16) = 2
DPP(16) = 1
D(17) = 3
DP(17) = 1
DPP(17) = 2
D(18) = 2
DP(18) = 1
```

```
      DPP(18) = 3
      D(19) = 2
      DP(19) = 3
      DPP(19) = 1
      D(20) = 1
      DP(20) = 2
      DPP(20) = 3
      D(21) = 2
      DP(21) = 1
      DPP(21) = 3
      D(22) = 3
      DP(22) = 2
      DPP(22) = 1
      D(23) = 1
      DP(23) = 3
      DPP(23) = 2
      D(24) = 2
      DP(24) = 1
      DPP(24) = 3
      D(25) = 1
      DP(25) = 2
      DPP(25) = 3
      D(26) = 2
      DP(26) = 3
      DPP(26) = 1
      D(27) = 3
      DP(27) = 1
      DPP(27) = 2

   ELSEIF((II.EQ.JJ).AND.(KK.GT.JJ)) THEN

      DEMAND(1) = DTEMP(21)
      DEMAND(2) = DTEMP(25)
      DEMAND(3) = DTEMP(12)
      DEMAND(4) = DTEMP(16)
      DEMAND(5) = DTEMP(3)
      DEMAND(6) = DTEMP(7)
      DEMAND(7) = DTEMP(20)
      DEMAND(8) = DTEMP(22)
      DEMAND(9) = DTEMP(24)
      DEMAND(10) = DTEMP(26)
      DEMAND(11) = DTEMP(11)
      DEMAND(12) = DTEMP(13)
      DEMAND(13) = DTEMP(15)
      DEMAND(14) = DTEMP(17)
      DEMAND(15) = DTEMP(2)
      DEMAND(16) = DTEMP(4)
      DEMAND(17) = DTEMP(6)
      DEMAND(18) = DTEMP(8)
      DEMAND(19) = DTEMP(19)
      DEMAND(20) = DTEMP(23)
      DEMAND(21) = DTEMP(10)
      DEMAND(22) = DTEMP(27)
      DEMAND(23) = DTEMP(14)
      DEMAND(24) = DTEMP(1)
      DEMAND(25) = DTEMP(18)
      DEMAND(26) = DTEMP(5)
      DEMAND(27) = DTEMP(9)
      DO 10 I = 1,27,2
            D(I) = 2
```

```
              DP(I) = 1
              DPP(I) = 3
10         CONTINUE
          DO 11 I = 2,26,2
             D(I) = 1
             DP(I) = 2
             DPP(I) = 3
11         CONTINUE

       ELSEIF((II.EQ.JJ).AND.(JJ.GT.KK)) THEN

          DEMAND(1) = DTEMP(9)
          DEMAND(2) = DTEMP(6)
          DEMAND(3) = DTEMP(8)
          DEMAND(4) = DTEMP(18)
          DEMAND(5) = DTEMP(5)
          DEMAND(6) = DTEMP(3)
          DEMAND(7) = DTEMP(7)
          DEMAND(8) = DTEMP(15)
          DEMAND(9) = DTEMP(17)
          DEMAND(10) = DTEMP(2)
          DEMAND(11) = DTEMP(4)
          DEMAND(12) = DTEMP(27)
          DEMAND(13) = DTEMP(14)
          DEMAND(14) = DTEMP(1)
          DEMAND(15) = DTEMP(12)
          DEMAND(16) = DTEMP(16)
          DEMAND(17) = DTEMP(24)
          DEMAND(18) = DTEMP(26)
          DEMAND(19) = DTEMP(11)
          DEMAND(20) = DTEMP(13)
          DEMAND(21) = DTEMP(23)
          DEMAND(22) = DTEMP(10)
          DEMAND(23) = DTEMP(21)
          DEMAND(24) = DTEMP(25)
          DEMAND(25) = DTEMP(20)
          DEMAND(26) = DTEMP(22)
          DEMAND(27) = DTEMP(19)
          DO 12 I = 1,13,2
             D(I) = 3
             DP(I) = 1
             DPP(I) = 2
12        CONTINUE
          DO 13 I = 2,14,2
             D(I) = 3
             DP(I) = 2
             DPP(I) = 1
13         CONTINUE

          DO 112 I = 15,27,2
             D(I) = 3
             DP(I) = 2
             DPP(I) = 1
112     CONTINUE
          DO 113 I = 16,26,2
             D(I) = 3
             DP(I) = 1
             DPP(I) = 2
113         CONTINUE
       ELSEIF((II.EQ.KK).AND.(JJ.GT.II)) THEN
```

```
      DEMAND(1)  = DTEMP(9)
      DEMAND(2)  = DTEMP(25)
      DEMAND(3)  = DTEMP(6)
      DEMAND(4)  = DTEMP(22)
      DEMAND(5)  = DTEMP(3)
      DEMAND(6)  = DTEMP(19)
      DEMAND(7)  = DTEMP(8)
      DEMAND(8)  = DTEMP(16)
      DEMAND(9)  = DTEMP(18)
      DEMAND(10)  = DTEMP(26)
      DEMAND(11)  = DTEMP(5)
      DEMAND(12)  = DTEMP(13)
      DEMAND(13)  = DTEMP(15)
      DEMAND(14)  = DTEMP(23)
      DEMAND(15)  = DTEMP(2)
      DEMAND(16)  = DTEMP(10)
      DEMAND(17)  = DTEMP(12)
      DEMAND(18)  = DTEMP(20)
      DEMAND(19)  = DTEMP(7)
      DEMAND(20)  = DTEMP(17)
      DEMAND(21)  = DTEMP(4)
      DEMAND(22)  = DTEMP(27)
      DEMAND(23)  = DTEMP(14)
      DEMAND(24)  = DTEMP(1)
      DEMAND(25)  = DTEMP(24)
      DEMAND(26)  = DTEMP(11)
      DEMAND(27)  = DTEMP(21)
      DO 14 I = 1,27,2
            D(I)  = 3
            DP(I)  = 1
            DPP(I)  = 2
14     CONTINUE
      DO 15 I = 2,26,2
         D(I)  = 1
         DP(I)  = 3
         DPP(I)  = 2
15     CONTINUE

    ELSEIF((II.EQ.KK).AND.(II.GT.JJ))THEN

      DEMAND(1)  = DTEMP(21)
      DEMAND(2)  = DTEMP(12)
      DEMAND(3)  = DTEMP(20)
      DEMAND(4)  = DTEMP(24)
      DEMAND(5)  = DTEMP(11)
      DEMAND(6)  = DTEMP(3)
      DEMAND(7)  = DTEMP(19)
      DEMAND(8)  = DTEMP(15)
      DEMAND(9)  = DTEMP(23)
      DEMAND(10)  = DTEMP(2)
      DEMAND(11)  = DTEMP(10)
      DEMAND(12)  = DTEMP(27)
      DEMAND(13)  = DTEMP(14)
      DEMAND(14)  = DTEMP(1)
      DEMAND(15)  = DTEMP(6)
      DEMAND(16)  = DTEMP(22)
      DEMAND(17)  = DTEMP(18)
      DEMAND(18)  = DTEMP(26)
      DEMAND(19)  = DTEMP(5)
      DEMAND(20)  = DTEMP(13)
```

```
         DEMAND(21) = DTEMP(17)
         DEMAND(22) = DTEMP(4)
         DEMAND(23) = DTEMP(9)
         DEMAND(24) = DTEMP(25)
         DEMAND(25) = DTEMP(8)
         DEMAND(26) = DTEMP(16)
         DEMAND(27) = DTEMP(7)
         DO 16 I = 1,13,2
            D(I) = 2
            DP(I) = 1
            DPP(I) = 3
16       CONTINUE
         DO 17 I = 2,14,2
            D(I) = 2
            DP(I) = 3
            DPP(I) = 1
17       CONTINUE

         DO 116 I = 15,27,2
            D(I) = 2
            DP(I) = 3
            DPP(I) = 1
116       CONTINUE
         DO 117 I = 16,26,2
            D(I) = 2
            DP(I) = 1
            DPP(I) = 3
117        CONTINUE

      ELSEIF((JJ.EQ.KK).AND.(II.GT.JJ)) THEN

         DEMAND(1) = DTEMP(9)
         DEMAND(2) = DTEMP(21)
         DEMAND(3) = DTEMP(8)
         DEMAND(4) = DTEMP(20)
         DEMAND(5) = DTEMP(7)
         DEMAND(6) = DTEMP(19)
         DEMAND(7) = DTEMP(6)
         DEMAND(8) = DTEMP(12)
         DEMAND(9) = DTEMP(18)
         DEMAND(10) = DTEMP(24)
         DEMAND(11) = DTEMP(5)
         DEMAND(12) = DTEMP(11)
         DEMAND(13) = DTEMP(17)
         DEMAND(14) = DTEMP(23)
         DEMAND(15) = DTEMP(4)
         DEMAND(16) = DTEMP(10)
         DEMAND(17) = DTEMP(16)
         DEMAND(18) = DTEMP(22)
         DEMAND(19) = DTEMP(3)
         DEMAND(20) = DTEMP(15)
         DEMAND(21) = DTEMP(2)
         DEMAND(22) = DTEMP(27)
         DEMAND(23) = DTEMP(14)
         DEMAND(24) = DTEMP(1)
         DEMAND(25) = DTEMP(26)
         DEMAND(26) = DTEMP(13)
         DEMAND(27) = DTEMP(25)
         DO 18 I = 1,27,2
            D(I) = 3
```

```
          DP(I) = 2
          DPP(I) = 1
18      CONTINUE
        DO 19 I = 2,26,2
          D(I) = 2
          DP(I) = 3
          DPP(I) = 1
19       CONTINUE

      ELSEIF((JJ.EQ.KK).AND.(II.LT.JJ)) THEN

        DEMAND(1) = DTEMP(25)
        DEMAND(2) = DTEMP(16)
        DEMAND(3) = DTEMP(22)
        DEMAND(4) = DTEMP(13)
        DEMAND(5) = DTEMP(26)
        DEMAND(6) = DTEMP(7)
        DEMAND(7) = DTEMP(19)
        DEMAND(8) = DTEMP(17)
        DEMAND(9) = DTEMP(23)
        DEMAND(10) = DTEMP(4)
        DEMAND(11) = DTEMP(10)
        DEMAND(12) = DTEMP(27)
        DEMAND(13) = DTEMP(14)
        DEMAND(14) = DTEMP(1)
        DEMAND(15) = DTEMP(8)
        DEMAND(16) = DTEMP(20)
        DEMAND(17) = DTEMP(18)
        DEMAND(18) = DTEMP(24)
        DEMAND(19) = DTEMP(5)
        DEMAND(20) = DTEMP(11)
        DEMAND(21) = DTEMP(15)
        DEMAND(22) = DTEMP(2)
        DEMAND(23) = DTEMP(9)
        DEMAND(24) = DTEMP(21)
        DEMAND(25) = DTEMP(6)
        DEMAND(26) = DTEMP(12)
        DEMAND(27) = DTEMP(3)
        DO 20 I = 1,13,2
          D(I) = 1
          DP(I) = 2
          DPP(I) = 3
20      CONTINUE
        DO 21 I = 2,14,2
          D(I) = 1
          DP(I) = 3
          DPP(I) = 2
21       CONTINUE

        DO 120 I = 15,27,2
          D(I) = 1
          DP(I) = 3
          DPP(I) = 2
120     CONTINUE
        DO 121 I = 16,26,2
          D(I) = 1
          DP(I) = 2
          DPP(I) = 3
121     CONTINUE
```

```
      ELSEIF((II.LT.JJ).AND.(JJ.LT.KK)) THEN

         DEMAND(1)  = DTEMP(25)
         DEMAND(2)  = DTEMP(16)
         DEMAND(3)  = DTEMP(7)
         DEMAND(4)  = DTEMP(22)
         DEMAND(5)  = DTEMP(26)
         DEMAND(6)  = DTEMP(13)
         DEMAND(7)  = DTEMP(17)
         DEMAND(8)  = DTEMP(4)
         DEMAND(9)  = DTEMP(8)
         DEMAND(10)  = DTEMP(19)
         DEMAND(11)  = DTEMP(23)
         DEMAND(12)  = DTEMP(10)
         DEMAND(13)  = DTEMP(27)
         DEMAND(14)  = DTEMP(14)
         DEMAND(15)  = DTEMP(1)
         DEMAND(16)  = DTEMP(18)
         DEMAND(17)  = DTEMP(5)
         DEMAND(18)  = DTEMP(9)
         DEMAND(19)  = DTEMP(20)
         DEMAND(20)  = DTEMP(24)
         DEMAND(21)  = DTEMP(11)
         DEMAND(22)  = DTEMP(15)
         DEMAND(23)  = DTEMP(2)
         DEMAND(24)  = DTEMP(6)
         DEMAND(25)  = DTEMP(21)
         DEMAND(26)  = DTEMP(12)
         DEMAND(27)  = DTEMP(3)
         DO 22 I = 1,27
               D(I) = 1
               DP(I) = 2
               DPP(I) = 3
22          CONTINUE

      ELSEIF((II.LT.KK).AND.(KK.LT.JJ)) THEN

         DEMAND(1)  = DTEMP(25)
         DEMAND(2)  = DTEMP(22)
         DEMAND(3)  = DTEMP(19)
         DEMAND(4)  = DTEMP(16)
         DEMAND(5)  = DTEMP(26)
         DEMAND(6)  = DTEMP(13)
         DEMAND(7)  = DTEMP(23)
         DEMAND(8)  = DTEMP(10)
         DEMAND(9)  = DTEMP(20)
         DEMAND(10)  = DTEMP(7)
         DEMAND(11)  = DTEMP(17)
         DEMAND(12)  = DTEMP(4)
         DEMAND(13)  = DTEMP(27)
         DEMAND(14)  = DTEMP(14)
         DEMAND(15)  = DTEMP(1)
         DEMAND(16)  = DTEMP(24)
         DEMAND(17)  = DTEMP(11)
         DEMAND(18)  = DTEMP(21)
         DEMAND(19)  = DTEMP(8)
         DEMAND(20)  = DTEMP(18)
         DEMAND(21)  = DTEMP(5)
         DEMAND(22)  = DTEMP(15)
         DEMAND(23)  = DTEMP(2)
```

```
      DEMAND(24)  =  DTEMP(12)
      DEMAND(25)  =  DTEMP(9)
      DEMAND(26)  =  DTEMP(6)
      DEMAND(27)  =  DTEMP(3)
      DO 23  I  =  1,27
            D(I)  =  1
            DP(I)  =  3
            DPP(I)  =  2
23       CONTINUE

   ELSEIF((JJ.LT.II).AND.(II.LT.KK)) THEN

      DEMAND(1)  =  DTEMP(21)
      DEMAND(2)  =  DTEMP(12)
      DEMAND(3)     DTEMP(3)
      DEMAND(4)     DTEMP(20)
      DEMAND(5)  .  DTEMP(24)
      DEMAND(6)  =  DTEMP(11)
      DEMAND(7)  =  DTEMP(15)
      DEMAND(8)  =  DTEMP(2)
      DEMAND(9)  =  DTEMP(6)
      DEMAND(10)  =  DTEMP(19)
      DEMAND(11)  =  DTEMP(23)
      DEMAND(12)  =  DTEMP(10)
      DEMAND(13)  =  DTEMP(27)
      DEMAND(14)  =  DTEMP(14)
      DEMAND(15)  =  DTEMP(1)
      DEMAND(16)  =  DTEMP(18)
      DEMAND(17)  =  DTEMP(5)
      DEMAND(18)  =  DTEMP(9)
      DEMAND(19)  =  DTEMP(22)
      DEMAND(20)  =  DTEMP(26)
      DEMAND(21)  =  DTEMP(13)
      DEMAND(22)  =  DTEMP(17)
      DEMAND(23)  =  DTEMP(4)
      DEMAND(24)  =  DTEMP(8)
      DEMAND(25)  =  DTEMP(25)
      DEMAND(26)  =  DTEMP(16)
      DEMAND(27)  =  DTEMP(7)
      DO 24  I  =  1,27
         D(I)  =  2
         DP(I)  =  1
         DPP(I)  =  3
24    CONTINUE

   ELSEIF((JJ.LT.KK).AND.(KK.LT.II)) THEN

      DEMAND(1)  =  DTEMP(21)
      DEMAND(2)  =  DTEMP(20)
      DEMAND(3)  =  DTEMP(19)
      DEMAND(4)  =  DTEMP(12)
      DEMAND(5)  =  DTEMP(24)
      DEMAND(6)  =  DTEMP(11)
      DEMAND(7)  =  DTEMP(23)
      DEMAND(8)  =  DTEMP(10)
      DEMAND(9)  =  DTEMP(22)
      DEMAND(10)  =  DTEMP(3)
      DEMAND(11)  =  DTEMP(15)
      DEMAND(12)  =  DTEMP(2)
      DEMAND(13)  =  DTEMP(27)
```

```
      DEMAND(14)  =  DTEMP(14)
      DEMAND(15)  =  DTEMP(1)
      DEMAND(16)  =  DTEMP(26)
      DEMAND(17)  =  DTEMP(13)
      DEMAND(18)  =  DTEMP(25)
      DEMAND(19)  =  DTEMP(6)
      DEMAND(20)  =  DTEMP(18)
      DEMAND(21)  =  DTEMP(5)
      DEMAND(22)  =  DTEMP(17)
      DEMAND(23)  =  DTEMP(4)
      DEMAND(24)  =  DTEMP(16)
      DEMAND(25)  =  DTEMP(9)
      DEMAND(26)  =  DTEMP(8)
      DEMAND(27)  =  DTEMP(7)
      DO 25 I = 1,27
         D(I) = 2
         DP(I) = 3
         DPP(I) = 1
25    CONTINUE

   ELSEIF((KK.LT.II).AND.(II.LT.JJ)) THEN

      DEMAND(1)  =  DTEMP(9)
      DEMAND(2)  =  DTEMP(6)
      DEMAND(3)  =  DTEMP(3)
      DEMAND(4)  =  DTEMP(8)
      DEMAND(5)  =  DTEMP(18)
      DEMAND(6)  =  DTEMP(5)
      DEMAND(7)  =  DTEMP(15)
      DEMAND(8)  =  DTEMP(2)
      DEMAND(9)  =  DTEMP(12)
      DEMAND(10)  =  DTEMP(7)
      DEMAND(11)  =  DTEMP(17)
      DEMAND(12)  =  DTEMP(4)
      DEMAND(13)  =  DTEMP(27)
      DEMAND(14)  =  DTEMP(14)
      DEMAND(15)  =  DTEMP(1)
      DEMAND(16)  =  DTEMP(24)
      DEMAND(17)  =  DTEMP(11)
      DEMAND(18)  =  DTEMP(21)
      DEMAND(19)  =  DTEMP(16)
      DEMAND(20)  =  DTEMP(26)
      DEMAND(21)  =  DTEMP(13)
      DEMAND(22)  =  DTEMP(23)
      DEMAND(23)  =  DTEMP(10)
      DEMAND(24)  =  DTEMP(20)
      DEMAND(25)  =  DTEMP(25)
      DEMAND(26)  =  DTEMP(22)
      DEMAND(27)  =  DTEMP(19)
      DO 26 I = 1,27
         D(I) = 3
         DP(I) = 1
         DPP(I) = 2
26    CONTINUE

   ELSEIF((KK.LT.JJ).AND.(JJ.LT.II)) THEN

      DEMAND(1)  =  DTEMP(9)
      DEMAND(2)  =  DTEMP(8)
      DEMAND(3)  =  DTEMP(7)
```

```
          DEMAND(4)  = DTEMP(6)
          DEMAND(5)  = DTEMP(18)
          DEMAND(6)  = DTEMP(5)
          DEMAND(7)  = DTEMP(17)
          DEMAND(8)  = DTEMP(4)
          DEMAND(9)  = DTEMP(3)
          DEMAND(10)  = DTEMP(15)
          DEMAND(11)  = DTEMP(2)
          DEMAND(12)  = DTEMP(27)
          DEMAND(13)  = DTEMP(14)
          DEMAND(14)  = DTEMP(1)
          DEMAND(15)  = DTEMP(26)
          DEMAND(16)  = DTEMP(13)
          DEMAND(17)  = DTEMP(16)
          DEMAND(18)  = DTEMP(25)
          DEMAND(19)  = DTEMP(12)
          DEMAND(20)  = DTEMP(24)
          DEMAND(21)  = DTEMP(11)
          DEMAND(22)  = DTEMP(23)
          DEMAND(23)  = DTEMP(10)
          DEMAND(24)  = DTEMP(22)
          DEMAND(25)  = DTEMP(21)
          DEMAND(26)  = DTEMP(20)
          DEMAND(27)  = DTEMP(19)
          DO 27 I = 1,27
             D(I) = 3
             DP(I) = 2
             DPP(I) = 1
 27       CONTINUE
       ENDIF

*    BEGIN TO FILL DEMAND NODES

       DO 45 I = 1,27
C         IF(DEMAND(I).EQ.0)GOTO45
          IF(S(D(I)) .GE. DEMAND(I))  THEN
             X(D(I),I) = DEMAND(I)
             S(D(I)) = S(D(I)) - X(D(I),I)
             DEMAND(I) = 0
             GOTO 45

          ELSEIF (S(D(I)) .GT. 0) THEN
             X(D(I),I)= S(D(I))
             S(D(I)) = S(D(I))- X(D(I),I)
             DEMAND(I) = DEMAND(I) - X(D(I),I)
             IF(DEMAND(I) .EQ. 0) GOTO45
          ENDIF

          IF (S(DP(I)) .GE. DEMAND(I)) THEN
             X(DP(I),I) = DEMAND(I)
             S(DP(I)) = S(DP(I)) - X(DP(I),I)
             DEMAND(I) = 0
             GOTO 45

          ELSEIF (S(DP(I)) .GT. 0) THEN
             X(DP(I),I)= S(DP(I))
             S(DP(I)) = S(DP(I)) - X(DP(I),I)
             DEMAND(I) = DEMAND(I) - X(DP(I),I)
             IF(DEMAND(I).EQ.0)GOTO 45
          ENDIF
```

```
         IF(S(DPP(I)) .GE. DEMAND(I)) THEN
            X(DPP(I),I) = DEMAND(I)
            S(DPP(I)) = S(DPP(I)) - X(DPP(I),I)
            DEMAND(I) = 0
            GOTO 45

         ELSEIF (S(DPP(I)) .GT.0) THEN
             X(DPP(I),I)= S(DPP(I))
             S(DPP(I)) = S(DPP(I)) - X(DPP(I),I)
             DEMAND(I) = DEMAND(I) - X(DPP(I),I)
         ENDIF
         IF (DEMAND(I).NE.0) PRINT*,'ERROR',DEMAND(I),I
 45    CONTINUE

*     REORDER FLOWS TO BE INTERPOLATED TO THE FINE GRID PROBLEM

      IF((II.EQ.JJ).AND.(JJ.EQ.KK)) THEN
         DO 30 L = 1,DIM
            X2H(L,1,1,1) = X(L,25)
            X2H(L,2,1,1) = X(L,22)
            X2H(L,3,1,1) = X(L,16)
            X2H(L,1,2,1) = X(L,23)
            X2H(L,2,2,1) = X(L,13)
            X2H(L,3,2,1) = X(L,4)
            X2H(L,1,3,1) = X(L,17)
            X2H(L,2,3,1) = X(L,5)
            X2H(L,3,3,1) = X(L,1)
            X2H(L,1,1,2) = X(L,24)
            X2H(L,2,1,2) = X(L,14)
            X2H(L,3,1,2) = X(L,6)
            X2H(L,1,2,2) = X(L,15)
            X2H(L,2,2,2) = X(L,26)
            X2H(L,3,2,2) = X(L,19)
            X2H(L,1,3,2) = X(L,7)
            X2H(L,2,3,2) = X(L,20)
            X2H(L,3,3,2) = X(L,10)
            X2H(L,1,1,3) = X(L,18)
            X2H(L,2,1,3) = X(L,8)
            X2H(L,3,1,3) = X(L,2)
            X2H(L,1,2,3) = X(L,9)
            X2H(L,2,2,3) = X(L,21)
            X2H(L,3,2,3) = X(L,11)
            X2H(L,1,3,3) = X(L,3)
            X2H(L,2,3,3) = X(L,12)
            X2H(L,3,3,3) = X(L,27)
 30   CONTINUE
      ELSEIF((II.EQ.JJ).AND.(KK.GT.JJ)) THEN
         DO 31 L = 1,DIM
            X2H(L,1,1,1) = X(L,24)
            X2H(L,2,1,1) = X(L,15)
            X2H(L,3,1,1) = X(L,5)
            X2H(L,1,2,1) = X(L,16)
            X2H(L,2,2,1) = X(L,26)
            X2H(L,3,2,1) = X(L,17)
            X2H(L,1,3,1) = X(L,6)
            X2H(L,2,3,1) = X(L,18)
            X2H(L,3,3,1) = X(L,27)
            X2H(L,1,1,2) = X(L,21)
            X2H(L,2,1,2) = X(L,11)
```

```
            X2H(L,3,1,2) = X(L,3)
            X2H(L,1,2,2) = X(L,12)
            X2H(L,2,2,2) = X(L,23)
            X2H(L,3,2,2) = X(L,13)
            X2H(L,1,3,2) = X(L,4)
            X2H(L,2,3,2) = X(L,14)
            X2H(L,3,3,2) = X(L,25)
            X2H(L,1,1,3) = X(L,19)
            X2H(L,2,1,3) = X(L,7)
            X2H(L,3,1,3) = X(L,1)
            X2H(L,1,2,3) = X(L,8)
            X2H(L,2,2,3) = X(L,20)
            X2H(L,3,2,3) = X(L,9)
            X2H(L,1,3,3) = X(L,2)
            X2H(L,2,3,3) = X(L,10)
            X2H(L,3,3,3) = X(L,22)
31      CONTINUE
        ELSEIF((II.EQ.JJ).AND.(JJ.GT.KK)) THEN
          DO 32 L = 1,DIM
            X2H(L,1,1,1) = X(L,14)
            X2H(L,2,1,1) = X(L,10)
            X2H(L,3,1,1) = X(L,6)
            X2H(L,1,2,1) = X(L,11)
            X2H(L,2,2,1) = X(L,5)
            X2H(L,3,2,1) = X(L,2)
            X2H(L,1,3,1) = X(L,7)
            X2H(L,2,3,1) = X(L,3)
            X2H(L,3,3,1) = X(L,1)
            X2H(L,1,1,2) = X(L,22)
            X2H(L,2,1,2) = X(L,19)
            X2H(L,3,1,2) = X(L,15)
            X2H(L,1,2,2) = X(L,20)
            X2H(L,2,2,2) = X(L,13)
            X2H(L,3,2,2) = X(L,8)
            X2H(L,1,3,2) = X(L,16)
            X2H(L,2,3,2) = X(L,9)
            X2H(L,3,3,2) = X(L,4)
            X2H(L,1,1,3) = X(L,27)
            X2H(L,2,1,3) = X(L,25)
            X2H(L,3,1,3) = X(L,23)
            X2H(L,1,2,3) = X(L,26)
            X2H(L,2,2,3) = X(L,21)
            X2H(L,3,2,3) = X(L,17)
            X2H(L,1,3,3) = X(L,24)
            X2H(L,2,3,3) = X(L,18)
            X2H(L,3,3,3) = X(L,12)
32      CONTINUE
        ELSEIF((II.EQ.KK).AND.(JJ.GT.II)) THEN
          DO 33 L = 1,DIM
            X2H(L,1,1,1) = X(L,24)
            X2H(L,2,1,1) = X(L,15)
            X2H(L,3,1,1) = X(L,5)
            X2H(L,1,2,1) = X(L,21)
            X2H(L,2,2,1) = X(L,11)
            X2H(L,3,2,1) = X(L,3)
            X2H(L,1,3,1) = X(L,19)
            X2H(L,2,3,1) = X(L,7)
            X2H(L,3,3,1) = X(L,1)
            X2H(L,1,1,2) = X(L,16)
            X2H(L,2,1,2) = X(L,26)
```

```
         X2H(L,3,1,2) = X(L,17)
         X2H(L,1,2,2) = X(L,12)
         X2H(L,2,2,2) = X(L,23)
         X2H(L,3,2,2) = X(L,13)
         X2H(L,1,3,2) = X(L,8)
         X2H(L,2,3,2) = X(L,20)
         X2H(L,3,3,2) = X(L,9)
         X2H(L,1,1,3) = X(L,6)
         X2H(L,2,1,3) = X(L,18)
         X2H(L,3,1,3) = X(L,27)
         X2H(L,1,2,3) = X(L,4)
         X2H(L,2,2,3) = X(L,14)
         X2H(L,3,2,3) = X(L,25)
         X2H(L,1,3,3) = X(L,2)
         X2H(L,2,3,3) = X(L,10)
         X2H(L,3,3,3) = X(L,22)
33    CONTINUE
      ELSEIF((II.EQ.KK).AND.(II.GT.JJ))THEN
         DO 34 L = 1,DIM
         X2H(L,1,1,1) = X(L,14)
         X2H(L,2,1,1) = X(L,10)
         X2H(L,3,1,1) = X(L,6)
         X2H(L,1,2,1) = X(L,22)
         X2H(L,2,2,1) = X(L,19)
         X2H(L,3,2,1) = X(L,15)
         X2H(L,1,3,1) = X(L,27)
         X2H(L,2,3,1) = X(L,25)
         X2H(L,3,3,1) = X(L,23)
         X2H(L,1,1,2) = X(L,11)
         X2H(L,2,1,2) = X(L,5)
         X2H(L,3,1,2) = X(L,2)
         X2H(L,1,2,2) = X(L,20)
         X2H(L,2,2,2) = X(L,13)
         X2H(L,3,2,2) = X(L,8)
         X2H(L,1,3,2) = X(L,26)
         X2H(L,2,3,2) = X(L,21)
         X2H(L,3,3,2) = X(L,17)
         X2H(L,1,1,3) = X(L,7)
         X2H(L,2,1,3) = X(L,3)
         X2H(L,3,1,3) = X(L,1)
         X2H(L,1,2,3) = X(L,16)
         X2H(L,2,2,3) = X(L,9)
         X2H(L,3,2,3) = X(L,4)
         X2H(L,1,3,3) = X(L,24)
         X2H(L,2,3,3) = X(L,18)
         X2H(L,3,3,3) = X(L,12)
34    CONTINUE
      ELSEIF((JJ.EQ.KK).AND.(II.GT.JJ)) THEN
         DO 35 L = 1,DIM
         X2H(L,1,1,1) = X(L,24)
         X2H(L,2,1,1) = X(L,21)
         X2H(L,3,1,1) = X(L,19)
         X2H(L,1,2,1) = X(L,15)
         X2H(L,2,2,1) = X(L,11)
         X2H(L,3,2,1) = X(L,7)
         X2H(L,1,3,1) = X(L,5)
         X2H(L,2,3,1) = X(L,3)
         X2H(L,3,3,1) = X(L,1)
         X2H(L,1,1,2) = X(L,16)
         X2H(L,2,1,2) = X(L,12)
```

```
            X2H(L,3,1,2)  = X(L,8)
            X2H(L,1,2,2)  = X(L,26)
            X2H(L,2,2,2)  = X(L,23)
            X2H(L,3,2,2)  = X(L,20)
            X2H(L,1,3,2)  = X(L,17)
            X2H(L,2,3,2)  = X(L,13)
            X2H(L,3,3,2)  = X(L,9)
            X2H(L,1,1,3)  = X(L,6)
            X2H(L,2,1,3)  = X(L,4)
            X2H(L,3,1,3)  = X(L,2)
            X2H(L,1,2,3)  = X(L,18)
            X2H(L,2,2,3)  = X(L,14)
            X2H(L,3,2,3)  = X(L,10)
            X2H(L,1,3,3)  = X(L,27)
            X2H(L,2,3,3)  = X(L,25)
            X2H(L,3,3,3)  = X(L,22)
 35     CONTINUE
        ELSEIF((JJ.EQ.KK).AND.(II.LT.JJ)) THEN
          DO 36 L = 1,DIM
            X2H(L,1,1,1)  = X(L,14)
            X2H(L,2,1,1)  = X(L,22)
            X2H(L,3,1,1)  = X(L,27)
            X2H(L,1,2,1)  = X(L,10)
            X2H(L,2,2,1)  = X(L,19)
            X2H(L,3,2,1)  = X(L,25)
            X2H(L,1,3,1)  = X(L,6)
            X2H(L,2,3,1)  = X(L,15)
            X2H(L,3,3,1)  = X(L,23)
            X2H(L,1,1,2)  = X(L,11)
            X2H(L,2,1,2)  = X(L,20)
            X2H(L,3,1,2)  = X(L,26)
            X2H(L,1,2,2)  = X(L,4)
            X2H(L,2,2,2)  = X(L,13)
            X2H(L,3,2,2)  = X(L,21)
            X2H(L,1,3,2)  = X(L,2)
            X2H(L,2,3,2)  = X(L,8)
            X2H(L,3,3,2)  = X(L,17)
            X2H(L,1,1,3)  = X(L,7)
            X2H(L,2,1,3)  = X(L,16)
            X2H(L,3,1,3)  = X(L,24)
            X2H(L,1,2,3)  = X(L,3)
            X2H(L,2,2,3)  = X(L,9)
            X2H(L,3,2,3)  = X(L,18)
            X2H(L,1,3,3)  = X(L,1)
            X2H(L,2,3,3)  = X(L,5)
            X2H(L,3,3,3)  = X(L,12)
 36     CONTINUE
        ELSEIF((II.LT.JJ).AND.(JJ.LT.KK)) THEN
          DO 37 L = 1,DIM
            X2H(L,1,1,1)  = X(L,15)
            X2H(L,2,1,1)  = X(L,23)
            X2H(L,3,1,1)  = X(L,27)
            X2H(L,1,2,1)  = X(L,8)
            X2H(L,2,2,1)  = X(L,17)
            X2H(L,3,2,1)  = X(L,24)
            X2H(L,1,3,1)  = X(L,3)
            X2H(L,2,3,1)  = X(L,9)
            X2H(L,3,3,1)  = X(L,18)
            X2H(L,1,1,2)  = X(L,12)
            X2H(L,2,1,2)  = X(L,21)
```

```
            X2H(L,3,1,2)  = X(L,26)
            X2H(L,1,2,2)  = X(L,6)
            X2H(L,2,2,2)  = X(L,14)
            X2H(L,3,2,2)  = X(L,22)
            X2H(L,1,3,2)  = X(L,2)
            X2H(L,2,3,2)  = X(L,7)
            X2H(L,3,3,2)  = X(L,16)
            X2H(L,1,1,3)  = X(L,10)
            X2H(L,2,1,3)  = X(L,19)
            X2H(L,3,1,3)  = X(L,25)
            X2H(L,1,2,3)  = X(L,4)
            X2H(L,2,2,3)  = X(L,11)
            X2H(L,3,2,3)  = X(L,20)
            X2H(L,1,3,3)  = X(L,1)
            X2H(L,2,3,3)  = X(L,5)
            X2H(L,3,3,3)  = X(L,13)
37    CONTINUE
      ELSEIF((II.LT.KK).AND.(KK.LT.JJ)) THEN
         DO 38 L = 1,DIM
            X2H(L,1,1,1)  = X(L,15)
            X2H(L,2,1,1)  = X(L,23)
            X2H(L,3,1,1)  = X(L,27)
            X2H(L,1,2,1)  = X(L,12)
            X2H(L,2,2,1)  = X(L,21)
            X2H(L,3,2,1)  = X(L,26)
            X2H(L,1,3,1)  = X(L,10)
            X2H(L,2,3,1)  = X(L,19)
            X2H(L,3,3,1)  = X(L,25)
            X2H(L,1,1,2)  = X(L,8)
            X2H(L,2,1,2)  = X(L,17)
            X2H(L,3,1,2)  = X(L,24)
            X2H(L,1,2,2)  = X(L,6)
            X2H(L,2,2,2)  = X(L,14)
            X2H(L,3,2,2)  = X(L,22)
            X2H(L,1,3,2)  = X(L,4)
            X2H(L,2,3,2)  = X(L,11)
            X2H(L,3,3,2)  = X(L,20)
            X2H(L,1,1,3)  = X(L,3)
            X2H(L,2,1,3)  = X(L,9)
            X2H(L,3,1,3)  = X(L,18)
            X2H(L,1,2,3)  = X(L,2)
            X2H(L,2,2,3)  = X(L,7)
            X2H(L,3,2,3)  = X(L,16)
            X2H(L,1,3,3)  = X(L,1)
            X2H(L,2,3.3)  = X(L,5)
            X2H(L,3,3,3)  = X(L,13)
38    CONTINUE
      ELSEIF((JJ.LT.II).AND.(II.LT.KK)) THEN
         DO 39 L = 1,DIM
            X2H(L,1,1,1)  = X(L,15)
            X2H(L,2,1,1)  = X(L,8)
            X2H(L,3,1,1)  = X(L,3)
            X2H(L,1,2,1)  = X(L,23)
            X2H(L,2,2,1)  = X(L,17)
            X2H(L,3,2,1)  = X(L,9)
            X2H(L,1,3,1)  = X(L,27)
            X2H(L,2,3,1)  = X(L,24)
            X2H(L,3,3,1)  = X(L,18)
            X2H(L,1,1,2)  = X(L,12)
            X2H(L,2,1,2)  = X(L,6)
```

```
            X2H(L,3,1,2)  = X(L,2)
            X2H(L,1,2,2)  = X(L,21)
            X2H(L,2,2,2)  = X(L,14)
            X2H(L,3,2,2)  = X(L,7)
            X2H(L,1,3,2)  = X(L,26)
            X2H(L,2,3,2)  = X(L,22)
            X2H(L,3,3,2)  = X(L,16)
            X2H(L,1,1,3)  = X(L,10)
            X2H(L,2,1,3)  = X(L,4)
            X2H(L,3,1,3)  = X(L,1)
            X2H(L,1,2,3)  = X(L,19)
            X2H(L,2,2,3)  = X(L,11)
            X2H(L,3,2,3)  = X(L,5)
            X2H(L,1,3,3)  = X(L,25)
            X2H(L,2,3,3)  = X(L,20)
            X2H(L,3,3,3)  = X(L,13)
39    CONTINUE
      ELSEIF((JJ.LT.KK).AND.(KK.LT.II)) THEN
         DO 40 L = 1,DIM
            X2H(L,1,1,1)  = X(L,15)
            X2H(L,2,1,1)  = X(L,12)
            X2H(L,3,1,1)  = X(L,10)
            X2H(L,1,2,1)  = X(L,23)
            X2H(L,2,2,1)  = X(L,21)
            X2H(L,3,2,1)  = X(L,19)
            X2H(L,1,3,1)  = X(L,27)
            X2H(L,2,3,1)  = X(L,26)
            X2H(L,3,3,1)  = X(L,25)
            X2H(L,1.1,2)  = X(L,8)
            X2H(L,2,1,2)  = X(L,6)
            X2H(L,3,1,2)  = X(L,4)
            X2H(L,1,2,2)  = X(L,17)
            X2H(L,2,2,2)  = X(L,14)
            X2H(L,3,2,2)  = X(L,11)
            X2H(L,1,3,2)  = X(L,24)
            X2H(L,2,3,2)  = X(L,22)
            X2H(L,3,3,2)  = X(L,20)
            X2H(L,1,1,3)  = X(L,3)
            X2H(L,2,1,3)  = X(L,2)
            X2H(L,3,1,3)  = X(L,1)
            X2H(L,1,2,3)  = X(L,9)
            X2H(L,2,2,3)  = X(L,7)
            X2H(L,3,2,3)  = X(L,5)
            X2H(L,1,3,3)  = X(L,18)
            X2H(L,2,3,3)  = X(L,16)
            X2H(L,3,3,3)  = X(L,13)
40    CONTINUE
      ELSEIF((KK.LT.II).AND.(II.LT.JJ)) THEN
         DO 41 L = 1,DIM
            X2H(L,1,1,1)  = X(L,15)
            X2H(L,2,1,1)  = X(L,8)
            X2H(L,3,1,1)  = X(L,3)
            X2H(L,1,2,1)  = X(L,12)
            X2H(L,2,2,1)  = X(L,6)
            X2H(L,3,2,1)  = X(L,2)
            X2H(L,1,3,1)  = X(L,10)
            X2H(L,2,3,1)  = X(L,4)
            X2H(L,3,3,1)  = X(L,1)
            X2H(L,1,1,2)  = X(L,23)
            X2H(L,2,1,2)  = X(L,17)
```

```
            X2H(L,3,1,2) = X(L,9)
            X2H(L,1,2,2) = X(L,21)
            X2H(L,2,2,2) = X(L,14)
            X2H(L,3,2,2) = X(L,7)
            X2H(L,1,3,2) = X(L,19)
            X2H(L,2,3,2) = X(L,11)
            X2H(L,3,3,2) = X(L,5)
            X2H(L,1,1,3) = X(L,27)
            X2H(L,2,1,3) = X(L,24)
            X2H(L,3,1,3) = X(L,18)
            X2H(L,1,2,3) = X(L,26)
            X2H(L,2,2,3) = X(L,22)
            X2H(L,3,2,3) = X(L,16)
            X2H(L,1,3,3) = X(L,25)
            X2H(L,2,3,3) = X(L,20)
            X2H(L,3,3,3) = X(L,13)
 41     CONTINUE
        ELSEIF((KK.LT.JJ).AND.(JJ.LT.II)) THEN
          DO 42 L = 1,DIM
            X2H(L,1,1,1) = X(L,14)
            X2H(L,2,1,1) = X(L,11)
            X2H(L,3,1,1) = X(L,9)
            X2H(L,1,2,1) = X(L,8)
            X2H(L,2,2,1) = X(L,6)
            X2H(L,3,2,1) = X(L,4)
            X2H(L,1,3,1) = X(L,3)
            X2H(L,2,3,1) = X(L,2)
            X2H(L,3,3,1) = X(L,1)
            X2H(L,1,1,2) = X(L,23)
            X2H(L,2,1,2) = X(L,21)
            X2H(L,3,1,2) = X(L,19)
            X2H(L,1,2,2) = X(L,16)
            X2H(L,2,2,2) = X(L,13)
            X2H(L,3,2,2) = X(L,10)
            X2H(L,1,3,2) = X(L,17)
            X2H(L,2,3,2) = X(L,7)
            X2H(L,3,3,2) = X(L,5)
            X2H(L,1,1,3) = X(L,27)
            X2H(L,2,1,3) = X(L,26)
            X2H(L,3,1,3) = X(L,25)
            X2H(L,1,2,3) = X(L,24)
            X2H(L,2,2,3) = X(L,22)
            X2H(L,3,2,3) = X(L,20)
            X2H(L,1,3,3) = X(L,18)
            X2H(L,2,3,3) = X(L,15)
            X2H(L,3,3,3) = X(L,12)
 42     CONTINUE
        ENDIF
        RETURN
        END

C       THE SUBROUTINE INTERP INTERPOLATES THE FLOWS FROM THE COARSE GRID
C       BACK TO THE FINE GRID.
C       THE ARRAY DH CONTAINS THE DEMANDS FOR THE FINE GRID DEMAND NODES.
C       THE ARRAYS DHTEMP, XHTEMP,AND STEMP ARE TEMPORARY ARRAYS USED
C       TO HOLD THE DEMAND, FLOW AND SUPPLY DURING THE INTERPOLATION
C       SCHEME. THE ARRAY X2H CONTAINS THE FLOWS FOR THE COARSE GRID
C       DEMAND NODES.  THE ARRAY XH CONTAINS THE INTERPOLATED FLOW FOR
C       THE FINE GRID.
```

```
      SUBROUTINE INTERP(DH,DHTEMP,XHTEMP,STEMP,X2H,XH,RC,R,DIM)
      INTEGER DIM,RC,R,L,LL
      REAL XH(DIM,R,R,R),DH(R,R,R),DHTEMP(DIM,DIM,DIM),XP2H(3,3,3,3),
     +X2H(DIM,RC,RC,RC),STEMP(DIM),XHTEMP(DIM,2*RC,2*RC,2*RC)
     +,IM,JM,KM,IP,JP,KP,IMJM,IMKM,IMJP,IMKP,JMKM,JMKP,JPKM,JPKP,
     +IPJM,IPJP,IPKM,IPKP,IMJMKM,IMJMKP,IMJPKM,IMJPKP,IPJMKM,IPJMKP,
     +IPJPKM,IPJPKP


      DO 1 K = 1,RC
         DO 1 J = 1,RC
            DO 1 I = 1,RC
               DO 1 L = 1,DIM
                  XHTEMP(L,2*I,2*J,2*K) = X2H(L,I,J,K)
  1   CONTINUE
      DO 2 K=1,3
         DO 2 J = 1,3
            DO 2 I = 1,3
               DO 2 L = 1,3
 2                XP2H(L,I,J,K)=0

C     RECOMPUTE DEMANDS FOR THE DEMAND NODES ON THE FINE GRID

      DO 5 K = 1,RC
         DO 5 J = 1,RC
            DO 5 I = 1,RC
               DO 6 KK = 1,3
                  DO 6 JJ = 1,3
                     DO 6 II = 1,3
                        DHTEMP(II,JJ,KK) = 0
  6            CONTINUE
               DO 7 LL = 1,DIM
                  STEMP(LL) = XHTEMP(LL,2*I,2*J,2*K)
  7            CONTINUE

               IM = .5
               IP = .5
               JM = .5
               JP = .5
               KM = .5
               KP = .5
               IMJM = .25
               IMJP = .25
               IPJM = .25
               IPJP = .25
               IMKM = .25
               IMKP = .25
               IPKM = .25
               IPKP = .25
               JMKM = .25
               JMKP = .25
               JPKM = .25
               JPKP = .25
               IMJMKM = .125
               IMJPKM = .125
               IMJMKP = .125
               IMJPKP = .125
               IPJMKM = .125
               IPJMKP = .125
               IPJPKM = .125
```

```
      IPJPKP = .125
C     THE IF STATEMENTS CHECK FOR EDGE AND CORNER DEMAND NODES.
      IF (I .EQ. 1) THEN
         IM = 1
         IMJM = .5
         IMJP = .5
         IMKM = .5
         IMKP = .5
         IMJPKM = .25
         IMJMKP = .25
         IMJPKP = .25
         IMJMKM = .25

      ELSEIF(I .EQ.RC) THEN
         IP = 1
         IPJM = .5
         IPJP = .5
         IPKM = .5
         IPKP = .5
         IPJPKM = .25
         IPJMKP = .25
         IPJPKP = .25
         IPJMKM = .25
      ENDIF
      IF (J .EQ. 1) THEN
         JM = 1
         IMJM = .5
         IPJM = .5
         JMKM = .5
         JMKP = .5
         IMJMKM = .25
         IPJMKP = .25
         IMJMKP = .25
         IPJMKM = .25
      ELSEIF (J .EQ.RC) THEN
         JP = 1
         IMJP = .5
         IPJP = .5
         JPKM = .5
         JPKP = .5
         IMJPKM = .25
         IMJPKP = .25
         IPJPKP = .25
         IPJPKM = .25
      ENDIF
      IF (K .EQ. 1)  THEN
         KM = 1
         IMKM = .5
         IPKM = .5
         JMKM = .5
         JPKM = .5
         IMJPKM = .25
         IMJMKM = .25
         IPJPKM = .25
         IPJMKM = .25
      ELSEIF (K .EQ.RC) THEN
         KP = 1
         IMKP = .5
```

```
     IPKP = .5
     JMKP = .5
     JPKP = .5
     IMJPKP = .25
     IMJMKP = .25
     IPJPKP = .25
     IPJMKP = .25
 ENDIF

     DHTEMP(2,2,2) = DH(2*I,2*J,2*K)
     DHTEMP(1,2,2) = IM*DH(2*I-1,2*J,2*K)
     DHTEMP(3,2,2) = IP*DH(2*I+1,2*J,2*K)
     DHTEMP(2,1,2) = JM*DH(2*I,2*J-1,2*K)
     DHTEMP(2,3,2) = JP*DH(2*I,2*J+1,2*K)
     DHTEMP(2,2,1) = KM*DH(2*I,2*J,2*K-1)
     DHTEMP(2,2,3) = KP*DH(2*I,2*J,2*K+1)

 IF ((I .EQ.1) .AND. (J .EQ. 1)) THEN
     IMJM = 1
     IMJMKP = .5
     IMJMKM = .5
 ELSEIF ((I .EQ.1) .AND. (J .EQ.RC)) THEN
     IMJP = 1
     IMJPKP = .5
     IMJPKM = .5
 ENDIF
 IF ((I .EQ.RC) .AND. (J .EQ. 1)) THEN
     IPJM = 1
     IPJMKP = .5
     IPJMKM = .5
 ELSEIF ((I .EQ.RC) .AND. (J .EQ.RC)) THEN
     IPJP = 1
     IPJPKP = .5
     IPJPKM = .5
 ENDIF
 IF ((I .EQ.1) .AND. (K .EQ. 1)) THEN
     IMKM = 1
     IMJMKM = .5
     IMJPKM = .5
 ELSEIF ((I .EQ.1) .AND. (K .EQ.RC)) THEN
     IMKP = 1
     IMJMKP = .5
     IMJPKP = .5
 ENDIF
 IF ((I .EQ.RC) .AND. (K .EQ. 1)) THEN
     IPKM = 1
     IPJMKM = .5
     IPJPKM = .5
 ELSEIF ((I .EQ.RC) .AND. (K .EQ.RC)) THEN
     IPKP = 1
     IPJMKP = .5
     IPJPKP = .5
 ENDIF
 IF ((J .EQ. 1) .AND. (K .EQ. 1)) THEN
     JMKM = 1
     IMJMKM = .5
     IPJMKM = .5
 ELSEIF ((J .EQ. 1) .AND. (K .EQ.RC)) THEN
     JMKP = 1
     IMJMKP = .5
```

```
            IPJMKP = .5
         ENDIF
         IF ((J .EQ.RC) .AND. (K .EQ. 1)) THEN
            JPKM = 1
            IMJPKM = .5
            IPJPKM = .5
         ELSEIF ((J .EQ.RC) .AND. (K .EQ.RC)) THEN
            JPKP = 1
            IMJPKP = .5
            IPJPKP = .5
         ENDIF

            DHTEMP(1,1,2) = IMJM*DH(2*I-1,2*J-1,2*K)
            DHTEMP(1,3,2) = IMJP*DH(2*I-1,2*J+1,2*K)
            DHTEMP(1,2,1) = IMKM*DH(2*I-1,2*J,2*K-1)
            DHTEMP(1,2,3) = IMKP*DH(2*I-1,2*J,2*K+1)
            DHTEMP(3,1,2) = IPJM*DH(2*I+1,2*J-1,2*K)
            DHTEMP(3,3,2) = IPJP*DH(2*I+1,2*J+1,2*K)
            DHTEMP(3,2,1) = IPKM*DH(2*I+1,2*J,2*K-1)
            DHTEMP(3,2,3) = IPKP*DH(2*I+1,2*J,2*K+1)
            DHTEMP(2,1,1) = JMKM*DH(2*I,2*J-1,2*K-1)
            DHTEMP(2,1,3) = JMKP*DH(2*I,2*J-1,2*K+1)
            DHTEMP(2,3,1) = JPKM*DH(2*I,2*J+1,2*K-1)
            DHTEMP(2,3,3) = JPKP*DH(2*I,2*J+1,2*K+1)

         IF ((I.EQ.1).AND.(J.EQ.1).AND.(K.EQ.1)) THEN
            IMJMKM = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.1).AND.(K.EQ.RC)) THEN
            IMJMKP = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.RC).AND.(K.EQ.1)) THEN
            IMJPKM = 1
         ELSEIF ((I.EQ.1).AND.(J.EQ.RC).AND.(K.EQ.RC)) THEN
            IMJPKP = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.1).AND.(K.EQ.1)) THEN
            IPJMKM = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.RC).AND.(K.EQ.1)) THEN
            IPJPKM = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.RC).AND.(K.EQ.RC)) THEN
            IPJPKP = 1
         ELSEIF ((I.EQ.RC).AND.(J.EQ.1).AND.(K.EQ.RC)) THEN
            IPJMKP = 1
         ENDIF

         DHTEMP(1,1,1)=IMJMKM*DH(2*I-1,2*J-1,2*K-1)
         DHTEMP(1,1,3)=IMJMKP*DH(2*I-1,2*J-1,2*K+1)
         DHTEMP(1,3,1)=IMJPKM*DH(2*I-1,2*J+1,2*K-1)
         DHTEMP(1,3,3)=IMJPKP*DH(2*I-1,2*J+1,2*K+1)
         DHTEMP(3,1,1)=IPJMKM*DH(2*I+1,2*J-1,2*K-1)
         DHTEMP(3,1,3)=IPJMKP*DH(2*I+1,2*J-1,2*K+1)
         DHTEMP(3,3,1)=IPJPKM*DH(2*I+1,2*J+1,2*K-1)
         DHTEMP(3,3,3)=IPJPKP*DH(2*I+1,2*J+1,2*K+1)

        IT = 2*I-1
        JT = 2*J-1
        KT = 2*K-1

C     SOLVE THE LOCAL 3X27 PROBLEM

        CALL LSOLVE(XP2H,DHTEMP,STEMP,DIM,IT,JT,KT)
           DO 25 L = 1,DIM
```

```
C     ACCUMULATE THE FLOW FOR EACH FINE GRID DEMAND NODE.

      XH(L,2*I-1,2*J-1,2*K-1)=XP2H(L,1,1,1)+XH(L,2*I-1,2*J-1,2*K-1)
      XH(L,2*I,2*J-1,2*K-1)=XP2H(L,2,1,1)+XH(L,2*I,2*J-1,2*K-1)
      XH(L,2*I+1,2*J-1,2*K-1)=XP2H(L,3,1,1)+XH(L,2*I+1,2*J-1,2*K-1)
      XH(L,2*I-1,2*J,2*K-1)=XP2H(L,1,2,1)+XH(L,2*I-1,2*J,2*K-1)
      XH(L,2*I,2*J,2*K-1)=XP2H(L,2,2,1)+XH(L,2*I,2*J,2*K-1)
      XH(L,2*I+1,2*J,2*K-1)=XP2H(L,3,2,1)+XH(L,2*I+1,2*J,2*K-1)
      XH(L,2*I-1,2*J+1,2*K-1)=XP2H(L,1,3,1)+XH(L,2*I-1,2*J+1,2*K-1)
      XH(L,2*I,2*J+1,2*K-1) = XP2H(L,2,3,1)+XH(L,2*I,2*J+1,2*K-1)
      XH(L,2*I+1,2*J+1,2*K-1) = XP2H(L,3,3,1)+XH(L,2*I+1,2*J+1,2*K-1)
      XH(L,2*I-1,2*J-1,2*K) = XP2H(L,1,1,2)+XH(L,2*I-1,2*J-1,2*K)
      XH(L,2*I,2*J-1,2*K) = XP2H(L,2,1,2)+XH(L,2*I,2*J-1,2*K)
      XH(L,2*I+1,2*J-1,2*K) = XP2H(L,3,1,2)+XH(L,2*I+1,2*J-1,2*K)
      XH(L,2*I-1,2*J,2*K) = XP2H(L,1,2,2)+XH(L,2*I-1,2*J,2*K)
      XH(L,2*I,2*J,2*K) = XP2H(L,2,2,2)+XH(L,2*I,2*J,2*K)
      XH(L,2*I+1,2*J,2*K) = XP2H(L,3,2,2)+XH(L,2*I+1,2*J,2*K)
      XH(L,2*I-1,2*J+1,2*K) = XP2H(L,1,3,2)+XH(L,2*I-1,2*J+1,2*K)
      XH(L,2*I,2*J+1,2*K) = XP2H(L,2,3,2)+XH(L,2*I,2*J+1,2*K)
      XH(L,2*I+1,2*J+1,2*K) = XP2H(L,3,3,2)+XH(L,2*I+1,2*J+1,2*K)
      XH(L,2*I-1,2*J-1,2*K+1) = XP2H(L,1,1,3)+XH(L,2*I-1,2*J-1,2*K+1)
      XH(L,2*I,2*J-1,2*K+1) = XP2H(L,2,1,3)+XH(L,2*I,2*J-1,2*K+1)
      XH(L,2*I+1,2*J-1,2*K+1) = XP2H(L,3,1,3)+XH(L,2*I+1,2*J-1,2*K+1)
      XH(L,2*I-1,2*J,2*K+1) = XP2H(L,1,2,3)+XH(L,2*I-1,2*J,2*K+1)
      XH(L,2*I,2*J,2*K+1) = XP2H(L,2,2,3)+XH(L,2*I,2*J,2*K+1)
      XH(L,2*I+1,2*J,2*K+1) = XP2H(L,3,2,3)+XH(L,2*I+1,2*J,2*K+1)
      XH(L,2*I-1,2*J+1,2*K+1) = XP2H(L,1,3,3)+XH(L,2*I-1,2*J+1,2*K+1)
      XH(L,2*I,2*J+1,2*K+1) = XP2H(L,2,3,3)+XH(L,2*I,2*J+1,2*K+1)
      XH(L,2*I+1,2*J+1,2*K+1)=XP2H(L,3,3,3)+XH(L,2*I+1,2*J+1,2*K+1)
 25           CONTINUE
  5   CONTINUE
      RETURN
      END

C     THE SUBROUTINE RELAX IMPROVES THE SOLUTION OBTAINED BY THE
C     THREE PREVIOUS SUBROUTINES.  THE SUBROUTINE CHECKS THE FLOW TO
C     EACH DEMAND FOR THE CHEAPEST SUPPLIER.  IF THE FLOW IS NOT FROM
C     THE CHEAPEST SUPPLIER, A DUMMY PROBLEM IS CREATED THAT IS OF THE
C     FORM AS THE ORIGINAL PROBLEM. THE ONLY SUPPLY AND DEMAND IS
C     FROM THE MISDIRECTED FLOW. THE ARRAY STEMP CONTAINS ALL THE
C     MISDIRECTED FLOW.  THE ARRAY XH CONTAINS THE FLOW FOR EACH FINE
C     GRID DEMAND NODE.  THE ARRAY DPTEMP CONTAINS THE MISFILLED DEMAND
C     NODE'S DEMAND.
C     THE RELAXATION TAKES PLACE ON THE GLOBAL GRID BY SOLVING SEVERAL
C     LOCAL 3X27 PROBLEMS.

      SUBROUTINE RELAX(STEMP,XH,RC,R,DIM)
      INTEGER DIM,R,RC,M,L,O,P,Q,S,C
      REAL STEMP(DIM),DPTEMP(3,3,3),XH(DIM,R,R,R),COST,
     +DTEMP(3,63,63,63),XHH(3,63,63,63),SUM,
     +XTEMP(3,3,3,3)
      CC=0
      DO 10 K=1,R
         DO 10 J = 1,R
            DO 10 I = 1,R
               DO 10 L = 1,DIM
                  DTEMP(L,I,J,K) = 0
                  XHH(L,I,J,K) = 0
 10   CONTINUE
      DO 15 K=1,3
```

```
      DO 15 J=1,3
         DO 15 I=1,3
            DPTEMP(I,J,K) =0
            DO 15 L=1,DIM
               XTEMP(L,I,J,K)=0
 15   CONTINUE

      STEMP(1) = 0
      STEMP(2) = 0
      STEMP(3) = 0
      COST = 0
      SUM = 0
      OPEN(UNIT = 12, FILE = 'FLOCOST DATA D',STATUS = 'OLD')

C     CHECK THE FLOW TO EACH DEMAND NODE TO VERIFY IF THE CHEAPEST
C     SUPPLIER WAS USED.  IF NOT CORRECT THE FLOW.

      DO 20 K = 1,R
         DO 20 J = 1,R
            DO 20 I = 1,R
               DO 20 L = 1,DIM
                  M = MIN(I,J,K)
                  IF(L .EQ. 1) THEN
                     IF(I.GT.M)THEN
                        DTEMP(L,I,J,K) = XH(L,I,J,K)+DTEMP(L,I,J,K)
                        STEMP(1) = XH(L,I,J,K)+ STEMP(1)
                        XH(L,I,J,K) = 0
                        COST = COST + DTEMP(L,I,J,K)*I
                     ENDIF
                     COST = COST + I*XH(L,I,J,K)
                  ELSEIF (L .EQ. 2) THEN
                     IF(J.GT.M)THEN
                        DTEMP(L,I,J,K) = XH(L,I,J,K)+DTEMP(L,I,J,K)
                        STEMP(2) = XH(L,I,J,K) + STEMP(2)
                        XH(L,I,J,K) = 0
                        COST = COST + DTEMP(L,I,J,K)*J
                     ENDIF
                     COST = COST + J*XH(L,I,J,K)
                  ELSE
                     IF(K.GT.M)THEN
                        DTEMP(L,I,J,K) = XH(L,I,J,K)+DTEMP(L,I,J,K)
                        STEMP(3) = XH(L,I,J,K)+ STEMP(3)
                        XH(L,I,J,K) = 0
                        COST = COST + DTEMP(L,I,J,K)*K
                     ENDIF
                     COST = COST + K*XH(L,I,J,K)
                  ENDIF
                 SUM = SUM + XH(L,I,J,K) + DTEMP(L,I,J,K)
 20   CONTINUE
      WRITE(12,13) COST,SUM,STEMP(1),STEMP(2),STEMP(3)
 12   FORMAT(2X,I2,2X,I2,2X,I2,2X,I2,2X,F9.2,2X,F9.2)
 13   FORMAT(2X,F12.2,2X,F12.2,2X,F8.2,2X,F8.2,2X,F8.2)

      DO 30 K = 1,RC
         DO 30 J = 1,RC
            DO 30 I =1,RC
               DO 30 L = 1,DIM

               DPTEMP(1,1,1) = DTEMP(L,2*I-1,2*J-1,2*K-1)
```

```
            DTEMP(L,2*I-1,2*J-1,2*K-1) = 0
            DPTEMP(2,1,1) = DTEMP(L,2*I,2*J-1,2*K-1)
            DTEMP(L,2*I,2*J-1,2*K-1) = 0
            DPTEMP(3,1,1) = DTEMP(L,2*I+1,2*J-1,2*K-1)
            DTEMP(L,2*I+1,2*J-1,2*K-1) = 0
            DPTEMP(1,2,1) = DTEMP(L,2*I-1,2*J,2*K-1)
            DTEMP(L,2*I-1,2*J,2*K-1) = 0
            DPTEMP(2,2,1) = DTEMP(L,2*I,2*J,2*K-1)
            DTEMP(L,2*I,2*J,2*K-1) = 0
            DPTEMP(3,2,1) = DTEMP(L,2*I+1,2*J,2*K-1)
            DTEMP(L,2*I+1,2*J,2*K-1) = 0
            DPTEMP(1,3,1) = DTEMP(L,2*I-1,2*J+1,2*K-1)
            DTEMP(L,2*I-1,2*J+1,2*K-1) = 0
            DPTEMP(2,3,1) = DTEMP(L,2*I,2*J+1,2*K-1)
            DTEMP(L,2*I,2*J+1,2*K-1) = 0
            DPTEMP(3,3,1) = DTEMP(L,2*I+1,2*J+1,2*K-1)
            DTEMP(L,2*I+1,2*J+1,2*K-1) = 0
            DPTEMP(1,1,2) = DTEMP(L,2*I-1,2*J-1,2*K)
            DTEMP(L,2*I-1,2*J-1,2*K) = 0
            DPTEMP(2,1,2) = DTEMP(L,2*I,2*J-1,2*K)
            DTEMP(L,2*I,2*J-1,2*K) = 0
            DPTEMP(3,1,2) = DTEMP(L,2*I+1,2*J-1,2*K)
            DTEMP(L,2*I+1,2*J-1,2*K) = 0
            DPTEMP(1,2,2) = DTEMP(L,2*I-1,2*J,2*K)
            DTEMP(L,2*I-1,2*J,2*K) = 0
            DPTEMP(2,2,2) = DTEMP(L,2*I,2*J,2*K)
            DTEMP(L,2*I,2*J,2*K) = 0
            DPTEMP(3,2,2) = DTEMP(L,2*I+1,2*J,2*K)
            DTEMP(L,2*I+1,2*J,2*K) = 0
            DPTEMP(1,3,2) = DTEMP(L,2*I-1,2*J+1,2*K)
            DTEMP(L,2*I-1,2*J+1,2*K) = 0
            DPTEMP(2,3,2) = DTEMP(L,2*I,2*J+1,2*K)
            DTEMP(L,2*I,2*J+1,2*K) = 0
            DPTEMP(3,3,2) = DTEMP(L,2*I+1,2*J+1,2*K)
            DTEMP(L,2*I+1,2*J+1,2*K) = 0
            DPTEMP(1,1,3) = DTEMP(L,2*I-1,2*J-1,2*K+1)
            DTEMP(L,2*I-1,2*J-1,2*K+1) = 0
            DPTEMP(2,1,3) = DTEMP(L,2*I,2*J-1,2*K+1)
            DTEMP(L,2*I,2*J-1,2*K+1) = 0
            DPTEMP(3,1,3) = DTEMP(L,2*I+1,2*J-1,2*K+1)
            DTEMP(L,2*I+1,2*J-1,2*K+1) = 0
            DPTEMP(1,2,3) = DTEMP(L,2*I-1,2*J,2*K+1)
            DTEMP(L,2*I-1,2*J,2*K+1) = 0
            DPTEMP(2,2,3) = DTEMP(L,2*I,2*J,2*K+1)
            DTEMP(L,2*I,2*J,2*K+1) = 0
            DPTEMP(3,2,3) = DTEMP(L,2*I+1,2*J,2*K+1)
            DTEMP(L,2*I+1,2*J,2*K+1) = 0
            DPTEMP(1,3,3) = DTEMP(L,2*I-1,2*J+1,2*K+1)
            DTEMP(L,2*I-1,2*J+1,2*K+1) = 0
            DPTEMP(2,3,3) = DTEMP(L,2*I,2*J+1,2*K+1)
            DTEMP(L,2*I,2*J+1,2*K+1) = 0
            DPTEMP(3,3,3) = DTEMP(L,2*I+1,2*J+1,2*K+1)
            DTEMP(L,2*I+1,2*J+1,2*K+1) = 0
            ITEMP = 2*I-1
            JTEMP = 2*J-1
            KTEMP = 2*K-1

C       SOLVE THE LOCAL 3X27 DUMMY PROBLEM TO CORRECT THE FLOW.

            CALL LSOLVE(XTEMP,DPTEMP,STEMP,DIM,ITEMP,JTEMP,KTEMP)
```

```
C       ACCUMULATE THE FLOW FOR EACH FINE GRID DEMAND NODE.
                 DO 140 C = 1,3
        XHH(C,2*I-1,2*J-1,2*K-1)=XTEMP(C,1,1,1)+XHH(C,2*I-1,2*J-1,2*K-1)
        XHH(C,2*I,2*J-1,2*K-1) = XTEMP(C,2,1,1)+XHH(C,2*I,2*J-1,2*K-1)
        XHH(C,2*I+1,2*J-1,2*K-1)=XTEMP(C,3,1,1)+XHH(C,2*I+1,2*J-1,2*K-1)
        XHH(C,2*I-1,2*J,2*K-1) = XTEMP(C,1,2,1)+XHH(C,2*I-1,2*J,2*K-1)
        XHH(C,2*I,2*J,2*K-1) = XTEMP(C,2,2,1)+XHH(C,2*I,2*J,2*K-1)
        XHH(C,2*I+1,2*J,2*K-1) = XTEMP(C,3,2,1)+XHH(C,2*I+1,2*J,2*K-1)
        XHH(C,2*I-1,2*J+1,2*K-1)=XTEMP(C,1,3,1)+XHH(C,2*I-1,2*J+1,2*K-1)
        XHH(C,2*I,2*J+1,2*K-1)=XTEMP(C,2,3,1)+XHH(C,2*I,2*J+1,2*K-1)
        XHH(C,2*I+1,2*J+1,2*K-1)=XTEMP(C,3,3,1)+XHH(C,2*I+1,2*J+1,2*K-1)
        XHH(C,2*I-1,2*J-1,2*K) = XTEMP(C,1,1,2)+XHH(C,2*I-1,2*J-1,2*K)
        XHH(C,2*I,2*J-1,2*K) = XTEMP(C,2,1,2)+XHH(C,2*I,2*J-1,2*K)
        XHH(C,2*I+1,2*J-1,2*K) = XTEMP(C,3,1,2)+XHH(C,2*I+1,2*J-1,2*K)
        XHH(C,2*I-1,2*J,2*K) = XTEMP(C,1,2,2)+XHH(C,2*I-1,2*J,2*K)
        XHH(C,2*I,2*J,2*K) = XTEMP(C,2,2,2)+XHH(C,2*I,2*J,2*K)
        XHH(C,2*I+1,2*J,2*K) = XTEMP(C,3,2,2)+XHH(C,2*I+1,2*J,2*K)
        XHH(C,2*I-1,2*J+1,2*K) = XTEMP(C,1,3,2)+XHH(C,2*I-1,2*J+1,2*K)
        XHH(C,2*I,2*J+1,2*K) = XTEMP(C,2,3,2)+XHH(C,2*I,2*J+1,2*K)
        XHH(C,2*I+1,2*J+1,2*K) = XTEMP(C,3,3,2)+XHH(C,2*I+1,2*J+1,2*K)
        XHH(C,2*I-1,2*J-1,2*K+1)=XTEMP(C,1,1,3)+XHH(C,2*I-1,2*J-1,2*K+1)
        XHH(C,2*I,2*J-1,2*K+1) = XTEMP(C,2,1,3)+XHH(C,2*I,2*J-1,2*K+1)
        XHH(C,2*I+1,2*J-1,2*K+1)=XTEMP(C,3,1,3)+XHH(C,2*I+1,2*J-1,2*K+1)
        XHH(C,2*I-1,2*J,2*K+1) = XTEMP(C,1,2,3)+XHH(C,2*I-1,2*J,2*K+1)
        XHH(C,2*I,2*J,2*K+1) = XTEMP(C,2,2,3)+XHH(C,2*I,2*J,2*K+1)
        XHH(C,2*I+1,2*J,2*K+1) = XTEMP(C,3,2,3)+XHH(C,2*I+1,2*J,2*K+1)
        XHH(C,2*I-1,2*J+1,2*K+1)=XTEMP(C,1,3,3)+XHH(C,2*I-1,2*J+1,2*K+1)
        XHH(C,2*I,2*J+1,2*K+1) = XTEMP(C,2,3,3)+XHH(C,2*I,2*J+1,2*K+1)
        XHH(C,2*I+1,2*J+1,2*K+1)=XTEMP(C,3,3,3)+XHH(C,2*I+1,2*J+1,2*K+1)

 140             CONTINUE
                 DO 35 O = 1,3
                    DO 35 P = 1,3
                       DO 35 Q = 1,3
                          DO 35 S = 1,3
                             IF ( O .EQ. 1)THEN
                                DPTEMP (P,Q,S) = 0
                             ENDIF
                             XTEMP(O,P,Q,S) = 0
 35              CONTINUE
 30     CONTINUE
        DO 45 K = 1,R
           DO 45 J = 1,R
              DO 45 I = 1,R
                 DO 45 L = 1,3
                    IF (XHH(L,I,J,K).NE.0) THEN
                      XH(L,I,J,K) = XHH(L,I,J,K) + XH(L,I,J,K)
                    ENDIF
 45     CONTINUE

C       COMPUTE THE COST

        COST = 0
        SUM = 0
        DO 50 K = 1,R
           DO 50 J = 1,R
              DO 50 I = 1,R
                 DO 50 L = 1,3
                    IF(L .EQ. 1) THEN
```

```
               COST = COST + I*XH(L,I,J,K)
            ELSEIF (L .EQ. 2) THEN
               COST = COST + J*XH(L,I,J,K)
            ELSE
               COST = COST + K*XH(L,I,J,K)
            ENDIF
            SUM = SUM + XH(L,I,J,K)
          IF(XH(L,I,J,K) .NE. 0)THEN
             WRITE(12,12)L,I,J,K,XH(L,I,J,K)
          ENDIF
50    CONTINUE
       WRITE(12,13) COST,SUM
      RETURN
      END
```

LIST OF REFERENCES

Balas, Egon, "Solution of Large-Scale Transportation Problems Through Aggregation," *Operations Research*, 13, 1965, pp. 82-84.

Bazaraa, M.S., Jarvis, J.J. and Sherali, H.D., *Linear Programming and Network Flows*, pp. 2-112 and 419-499, John Wiley and Sons, 1990.

Bradley, G.H., Brown, G.G., and Graves, G.W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, v. 24, pp. 1-31, 1 September 1977.

Briggs, W.L., *A Multigrid Tutorial*, pp. 1-63, Society for Industrial and Applied Mathematics, 1987.

Cavanaugh, Kevin, *Multigrid Approach to the Long Transportation Problem*, Master's Thesis, Naval Postgraduate School, September 1992.

Hadley, G., *Linear Programming*, pp. 273-323, Addison-Wesley Publishing Co., 1962.

Kaminsky, R., *Multilevel Solution of the Long Transportation Problem*, Master's Thesis, Weizmann Institute of Science, February 1989.

McCormick, S., *Multilevel Adaptive Methods for Partial Differential Equations*, Society for Industrial Applied Mathematics, 1989.

Rosen, Kenneth H., *Discrete Mathematics and its Applications*, pp. 312-317, McGraw-Hill, Inc., 1991.

Wesseling, P., *An Introduction to Multigrid Methods*, pp. 1-77, John Wiley and Sons, 1992.

Zipkin, Paul H., *Aggregation in Linear Programming*, Ph.D. Dissertation, Yale University, 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Van Emden Henson, Code MA/Hv Naval Postgraduate School Monterey, California 93943-5002	2
4.	Craig W. Rasmussen, Code MA/Ra Naval Postgraduate School Monterey, California 93943-5002	2
5.	LCDR Kevin Cavanaugh, USCG 1 Norman Ct Ferry Gables, Connecticut 06335	1
6.	LT Annette P. Cornett, USN 1606 Kickapoo Road Pueblo, Colorado 81001	2